I LIBRI di ioPROGRAMMO

Un manuale, facile, veloce per usare subito i servizi disponibili su Internet e crearne di propri

Guida Pratica

Web Services

Ivan Venuti

EDIZIONI MASTER

i libri di
ioPROGRAMMO

GUIDA PRATICA

WEB SERVICES

Ivan Venuti

EDIZIONI
MASTER

INTRODUZIONE

L'informatica è una scienza in continua evoluzione. Evolvono i linguaggi di programmazione, passando a linguaggi sempre più astratti - e per questo più vicini al modo di "rappresentare" le informazioni da parte di persone, piuttosto che di elaboratori - ma evolvono anche le architetture dei sistemi informativi. Questo libro affronta una delle più moderne architetture apparse gli ultimi anni: i servizi Web o, all'inglese, Web Services (spesso abbreviati in WS) e l'emergente SOA (Service Oriented Architecture). I Web Services sono talmente astratti che non si preoccupano nemmeno di quali linguaggi di programmazione li implementano, garantendo (almeno nella teoria) una completa interoperabilità tra linguaggi e piattaforme software diverse e ponendo dei vincoli solo sui formati di comunicazione tra gli attori interessati. Però per comprendere a fondo l'architettura, benché essa sia svincolata da un linguaggio di programmazione, è indispensabile realizzare esempi concreti. Ecco allora l'idea iniziale del libro: presentare, per ciascun linguaggio di programmazione (tra quelli maggiormente diffusi), una implementazione di (almeno) un client per accedere ai servizi Web esistenti e mostrare in alcuni casi come realizzare anche la parte server. Lo scopo del libro è essenzialmente pratico. Per raggiungere tale scopo verranno sempre mostrati esempi completi, adatti ad essere utilizzati fin da subito per comprendere le nozioni teoriche che verranno proposte nei primi capitoli. Avere a disposizione numerosi esempi realizzati nei più vari linguaggi permetterà anche una facile e veloce comparazione del diverso "potere espressivo" dei linguaggi presentati ma, soprattutto, aiuterà ad analizzare le problematiche connesse alla creazione di servizi Web ponendo come problematica centrale l'interoperabilità. Mi auguro che chiunque possa trovare almeno una parte del libro utile e adatta alle proprie esigenze e che il resto dei capitoli possa rappresentare comunque una lettura interessante. Diventerà evidente, man mano che si procederà nella lettura, quali siano le caratteristiche che fanno dei Web Services una tra le tecnologie più promettenti e destinata ad essere una sicura protagonista del prossimo futuro. Scrivere esempi con numerosi tool e

PuntoInformatico | Libri

linguaggi di programmazione rischiava di essere un lavoro estremamente lungo e, per forza di cose, dai risultati poco "in linea" con la filosofia propria degli strumenti che utilizzo di rado. È per questo che mi sono rivolto ad alcune persone, ciascuna delle quali mi ha aiutato a realizzare quanto mi ero prefissato: un sentito ringraziamento all'amico Filippo Costalli, autore degli esempi in PHP, e all'amico Andrea Maestrutti (dree) a cui si devono gli esempi scritti in Perl. Grazie ad entrambi anche per i preziosi suggerimenti e la generosa disponibilità, dimostrata da sempre e confermata anche in questa circostanza. Grazie anche alla comunity di http://www.perl.it (in particolare a Stefano Rodighiero che ha realizzato una GUI multipiattaforma per il client Perl) che hanno contribuito agli esempi sul Perl e a quanti, nelle diverse mailing list, hanno potuto darmi ottimi consigli e suggerimenti per superare alcuni ostacoli. Chiunque vorrà farmi pervenire le proprie impressioni e i propri commenti riguardo ai contenuti del libro, o vorrà segnalarmi eventuali inesattezze, proposte di miglioramento o suggerimenti, è il benvenuto e non mancherò di tenere traccia, sul sito http://ivenuti.altervista.org, di tutte le segnalazioni più significative.
E ora… Web Services per tutti i gusti!

Ivan Venuti, *ivanvenuti@yahoo.it*

Biografia: Ivan Venuti è laureato in Informatica e lavora come analista programmatore in un'azienda che si occupa di sviluppare programmi con tecnologia J2EE. La passione per la programmazione lo ha portato ad approfondire nuove tecnologie emergenti. Scrive abitualmente articoli per alcune tra le maggiori riviste di informatica. Sul suo sito personale, raggiungibile all'indirizzo *http://ivenuti.altervista.org*, sono messe a disposizione notizie e informazioni aggiornate sulla sua attività professionale.

INTRODUZIONE ALLE ARCHITETTURE DISTRIBUITE

Negli anni le applicazioni si sono evolute e si è evoluto il modo in cui esse sono strutturate: sono passate da architetture locali ad architetture distribuite. Esempi di architetture locali sono quelle in cui l'applicazione principale e le componenti "secondarie", utilizzate da essa (altre applicazioni, ma anche database, file system, e così via) risiedono tutte sulla stessa macchina. Questo tipo di architettura è efficiente e semplice, la comunicazione tra le diverse applicazioni e componenti avviene in maniera veloce e non presenta criticità. Purtroppo non ottimizza l'uso delle risorse, in quanto esse sono tutte dedicate a soddisfare il lavoro di un singolo utente. Inoltre i dati a disposizione non sono in alcun modo sincronizzati con altre postazioni di lavoro e questo può portare ad una proliferazione e ridondanza di dati.
Un'architettura distribuita è un'architettura dove le diverse applicazioni possono risiedere su nodi diversi, messi in comunicazione da un qualche tipo di rete (locale, nel caso di singoli edifici cablati, o geografica quando i nodi sono dislocati su superfici più ampie). In questo caso si complica la comunicazione tra le applicazioni ma si hanno diversi vantaggi, primo tra tutti quello di dare ad una o più applicazioni coinvolte la possibilità di essere utilizzate in concorrenza da più utenti o di centralizzare i dati a cui accedono le componenti periferiche.
Un esempio di architettura distribuita particolarmente diffusa è rendere centralizzato il database e accedere da applicazioni distribuite su nodi diversi: in questo modo i dati vengono gestiti in maniera organica e si possono fattorizzare le informazioni di un'intera organizzazione. Anche i file system possono essere condivisi, con i medesimi vantaggi dei database.
Ma anche le applicazioni possono essere realizzate con un'ottica distribuita: un'unica applicazione centralizzata offre dei servizi ad una serie di client che si occupano di comunicare con

PuntoInformatico | Libri

essa attraverso un'interfaccia, che può essere un semplice front-end senza logica oppure un'applicazione vera e propria che si avvale di funzionalità specifiche di una seconda applicazione distribuita.
In base al "grado" di distribuzione si possono avere architetture che prendono nomi particolari. Vediamo le più comuni

1.1 CLIENT-SERVER

L'architettura distribuita più comune è a due livelli: un fornitore del servizio (server) e un utilizzatore (client). Il client può essere un semplice visualizzatore (lo è il browser per le pagine HTML) o può includere un minimo di funzionalità (è il caso del browser con interprete JavaScript).

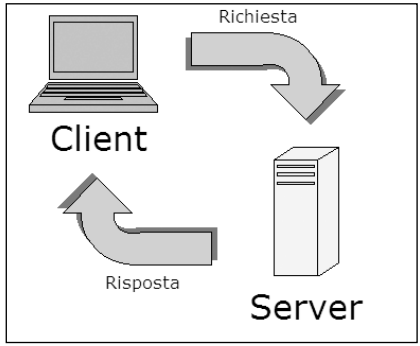

Figura 1.1: una tipica architettura client/server.

In pratica si tenta di creare un server molto potente che esegue tutte le operazioni (o quasi) e un client molto semplice che ha il compito di visualizzarle ed eventualmente di mandare dei comandi di modifica al server. Esempi tipici di quest'architettura sono siti Web, dal contenuto statico o dinamico, o applicazioni realizzate per essere installate su singole macchine e che fan-

no uso della maggior parte dei dati reperendoli da un server centrale (è il caso di applet Java, applicazioni remote con i dati centralizzati e così via).
Questa architettura prevede che il server sia implementato come un servizio unico, mastodontico.
È una soluzione che si ritrova facilmente in tutta una serie di applicazioni sviluppate in linguaggi come il Visual Basic.
Anche se non è il linguaggio a forzare un'architettura piuttosto che un'altra, è vero che certi linguaggi sono pensati per realizzare alcune architetture piuttosto che altre. Se il server diviene a più livelli "logici", si passa ad architetture più evolute.

1.2 TRE LIVELLI (O PIÙ)

Il server, per fornire i servizi, può anche delegare ad altre entità parte della gestione dei dati o dell'elaborazione. La prima architettura a più livelli prevedeva che il server fosse composto da una parte dedicata alla gestione dei dati persistenti e da un'altra dedicata alla logica e trasformazione dei dati.
A partire da questa si sono evolute architetture sempre più complesse, dove gli "strati" logici in cui è scomposto il server aumentano e ciascuno strato si occupa di una particolare gestione: chi dei dati e della persistenza, chi delle operazioni (questa soluzione è detta anche "logica di business"), chi della trasformazione in un formato adatto al client (infatti i client sono divenuti via via sempre più disomogenei, sia in termini di prestazioni che di dati che potevano accettare: si pensi a browser Web, palmari, telefoni, televisori digitali etc.).
Con l'affermarsi dei pattern architetturali si è andata delineando una serie di indicazioni su come suddividere le applicazioni server: attualmente va per la maggiore il pattern architetturale MVC (model view controller).
In ambiente J2EE la parte di model viene gestita da una o più

PuntoInformatico Libri

servlet, quella di controller da Java Bean o EJB (Enterprise JavaBean) mentre la parte di view "canonica" viene realizzata attraverso le JSP (Java Server Pages).

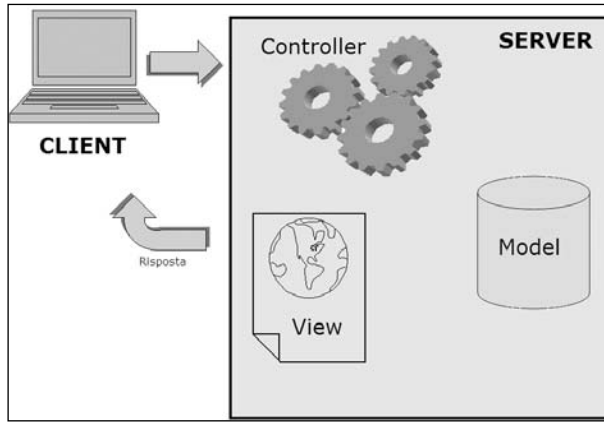

Figura 1.2: una tipica architettura client/server.

1.3 SERVICE ORIENTED ARCHITECTURE (SOA)

Quando si sono iniziate a delineare delle architetture completamente distribuite (ovvero architetture ove i diversi strati, dati, business e presentazione, non erano più all'interno di un'unica macchina o rete locale, ma erano distribuite geograficamente) c'era la necessità di stabilire regole di "comportamento" tra i diversi attori. Nascono così delle architetture complesse da gestire e vincolate in maniera pesante dal protocollo di comunicazione e dalle tecnologie coinvolte. Queste tecnologie cercano di fornire specifiche per rappresentare gli oggetti che vengono inviati e ricevuti, specificano il loro ciclo di vita e stabiliscono veri e propri pattern di comunicazione; tra le tecnologie più famose ricordiamo CORBA *(Common Object Request Broker Architecture, standard multipiattaforma per linguaggi orienta-*

ti agli oggetti), RMI *(Remote Method Invocation, specifico del mondo Java)* e DCOM *(Distrubuted COM, tecnologia proprietaria di Microsoft)* [AAVV, 2001].
Ben presto si è cercato un linguaggio di comunicazione davvero universale e che non soffrisse di tutta la complicata infrastruttura delle tecnologie precedenti; ecco affacciarsi i primi esempi di RPC (Remote Procedure Call) con uso di XML. Questo permette di invocare un servizio remoto utilizzando l'XML come formato di rappresentazione e di ricevere un risultato, senza preoccuparsi di problemi di persistenza o di ciclo di vita degli oggetti.
Il passo successivo è stato cercare di rendere standard le tecnologie coinvolte: nascono i primi Web Services. Queste tecnologie cercano di condividere singole unità operative (oggetti, funzionalità a grana fine, ovvero funzionalità semplici ma che coinvolgevano oggetti distribuiti su più macchine). La vera "rivoluzione" è stata quando si è iniziato a pensare ad uno scenario in cui non erano i singoli componenti ad essere condivisi in rete, ma erano i servizi che ogni componente riesce a fornire. Tali servizi sono, possibilmente, a grana grossa, ovvero espongono delle funzionalità complesse e non banali. In questo scenario non ci sono più un unico fornitore e un fruitore, ma i fruitori sono, a loro volta, fornitori di servizi più complessi. Si può pensare che i diversi servizi eseguano solo delle funzionalità specifiche, complete ma "atomiche". Sono dei "mattoni" di elaborazione che possono essere a loro volta combinati per fornire "mattoni" più complessi e, tutti insieme, possono concorrere ad espletare diverse funzionalità. L'architettura non si basa più su "ruoli" (cliente/fornitore) ma su servizi che ognuno espone e offre agli altri attraverso opportuni registri che mantengono le informazioni sui servizi pubblicati. Questa architettura prende il nome di *Service Oriented Architecture* (SOA) e la base su cui è stata costruita sono i Web Services e le tecnologie che li realizzano.

PuntoInformatico | Libri

Benché nel caso più semplice (**Figura 1.3**) essa sia una semplice ri-elaborazione dell'architettura client/server, nel caso generale (**Figura 1.4**) essa mostra la sua complessità e necessita di un'ar-

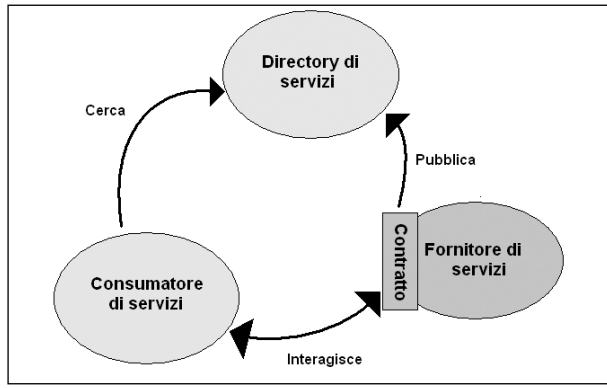

Figura 1.3: SOA (caso di un utilizzatore, un fornitore ed un registro)

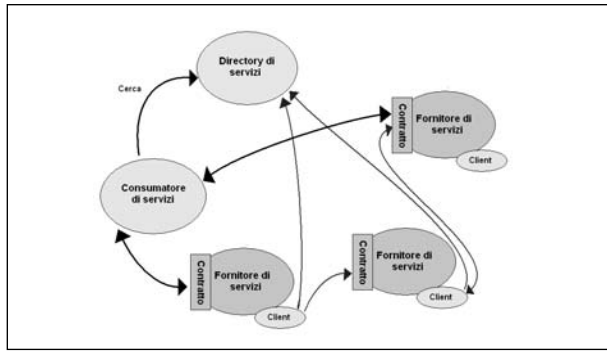

Figura 1.4: SOA (caso generale)

ticolata e complessa azione di management.
Si noti che i Web Services da soli non permettono lo sviluppo di architetture SOA, ma ne sono la base. È un po' come dire che i mattoni da soli non permettono di costruire le case, ma

sono i componenti senza cui non si potrebbero costruire (ma non ci dilunghiamo qui sui vari metodi costruttivi). In questo libro si vedranno le basi per costruire Web Services e si mostreranno gli scenari d'uso in cui essi possono essere utilizzati in un'ottica SOA. SOA si prefigge di:

1. rendere riutilizzabili i singoli servizi in diversi contesti;
2. non vincolarsi all'uso di un'architettura di riferimento, ma di vincolare le modalità di comunicazione (astrazione che passa dall'implementazione ai dati e alle loro modalità di fruizione);
3. far diventare ogni servizio indispensabile unicamente in quanto fornisce certe capacità al sistema; è possibile sostituirlo in qualsiasi momento con un servizio dalle capacità analoghe (ma più economico, efficiente, generale o dotato di altre caratteristiche desiderabili);
4. garantire l'esecuzione del servizio in maniera affidabile; si pensi un po' a ciò che avviene per la rete Internet, in cui anche la mancanza di alcuni attori non blocca l'intera rete, che rimane in grado di funzionare, pur se con prestazioni ridotte.

PuntoInformatico | Libri

1.4 COSA SONO I WEB SERVICES?

Un Web Service è un insieme di standard di comunicazione che permettono a diverse applicazioni di scambiarsi dati e servizi applicativi. Lo scenario ipotizzato è quello di un'applicazione che necessita, per espletare le sue funzioni, di una serie di servizi. Si vogliono reperire altrove questi servizi invece di svilupparli all'interno dell'applicazione stessa. Nel libro forniremo un esempio completo di realizzazione di un Web Service e di diversi client che ne fanno uso. Per esempio si supponga che l'applicazione gestisca l'ordine di un nuovo PC assemblato attraverso il Web. La sua fun-

zionalità è quella di ricevere richieste da parte di un utente che, dopo essersi autenticato, ordina una determinata configurazione hardware e software. L'applicazione stessa poi ordina i pezzi e il software utilizzando servizi esposti da fornitori esterni. Nel farlo si vorrebbe far sì che non acceda sempre e solo ad un unico servizio, ma valuti (ordine per ordine) la convenienza di un fornitore rispetto ad un altro e ordini direttamente i pezzi necessari dal miglior offerente. Una volta reperiti i pezzi e ordinati (sempre in tempo reale), l'applicazione potrebbe chiedere all'utente di effettuare il pagamento attraverso carta di credito: il pagamento vero e proprio (che comprende la verifica della correttezza dei dati della carta di credito e tutte le eventuali operazioni ausiliarie necessarie per il suo espletamento) potrebbe essere demandato ad un ulteriore sevizio Web. In questo contesto si noti come:

1. i diversi servizi sono dislocati geograficamente;
2. è necessario definire le modalità di reperimento dei servizi Web disponibili per determinate funzionalità;
3. è necessario prevedere un formato per il "dialogo" con i diversi servizi utilizzati.

Si vedrà come le tecnologie che permettono l'uso dei servizi Web siano pensate e studiate al fine di garantire una forte interoperabilità indipendentemente dalle tecnologie utilizzate per realizzare i diversi servizi. Vanno anche identificate le possibili informazioni "riservate" a cui si deve prestare attenzione affinché non possano essere oggetto di attacchi da parte di malintenzionati (ordini fasulli, richieste di pagamento non autorizzate e così via).

1.5 PERCHÉ USARE I WEB SERVICES?

I Web Services, come si è visto, non sono stati la prima for-

malizzazione di un'architettura distribuita. Rispetto alle precedenti però quest'architettura offre dei vantaggi che l'hanno fatta preferire rispetto alle precedenti:

1) semplicità della specifica;
2) utilizzo di tecnologie standard;
3) ampio consenso da parte di diverse aziende (Microsoft, Sun, IBM, tanto per citare quelle più significative);
4) presenza di tool che aiutano enormemente la creazione di nuovi servizi e la fruizione dei servizi esistenti.

Questi vantaggi hanno portato alla sua adozione in diversi contesti. Come accade spesso, avere a disposizione numerose implementazioni, sia di servizi che di strumenti per realizzarli, porta la specifica a diventare uno "standard di fatto" e a imporsi sul mercato. Questo è accaduto per i Web Services di cui, ad oggi, esistono almeno un centinaio di tool per agevolarne lo sviluppo.
Questi tool sono disponibili per le più svariate piattaforme software e per tutti i più diffusi linguaggi di programmazione. Questo libro è proprio una panoramica, seppur parziale, dei tool e del loro uso in situazioni concrete.

PuntoInformatico | Libri

1.6 SEMPLICITÀ DELLA SPECIFICA

Più una specifica è semplice, minore è il tempo che trascorre tra il suo studio e la realizzazione di progetti concreti. Questo vale sia in termini di servizi ma anche, e soprattutto, per tool di sviluppo. Se la tecnologia è complessa solo in certi ambiti lavorativi è pensabile una sua adozione in ambiti dove i tempi di sviluppo dei servizi sono talmente lunghi per cui la fase di studio e sperimentazione può essere di molti mesi. In altri casi, il più delle volte, o esistono società che hanno interesse a conseguire una specializzazione nel contesto architetturale

oppure ci si rivolge a terze parti o, se i vincoli architetturali lo permettono, si tentano strade alternative. Le tecnologie che stanno alla base dei Web Services sono molto semplici, intuitive e con poche regole di base. Purtroppo questo non resta valido quando le applicazioni crescono di complessità e si vogliono costruire architetture SOA: ultimamente sono nate moltissime specifiche per i più disparati utilizzi specifici e questo sta complicando le specifiche, mettendo in allarme gli sviluppatori. Per fortuna l'esistenza di tool si sviluppo sempre più sofisticati agevola la scrittura delle parti più ripetitive e permette di essere produttivi da subito anche per applicazioni con una certa complessità.

1.7 UTILIZZO DI TECNOLOGIE STANDARD

Aldilà della semplicità della specifica tutte le tecnologie alla base dei Web Services sono standard (essenzialmente XML per la rappresentazione, XML Schema per la validazione e Http per il trasporto, anche se, come vedremo, il trasporto si può basare su qualsiasi altro protocollo). Questo riuso di tecnologie esistenti, collaudate e adottate da un gran numero di sviluppatori, contribuisce a rendere la tecnologia accessibile. L'XML come scelta di base permette, inoltre, anche ad una persona di "guardare" i messaggi e di intuirne il significato (anche se essi sono stati realizzati per essere utilizzati da applicazioni).

1.8 AMPIO CONSENSO DA PARTE DI DIVERSE AZIENDE

Microsoft è stata la prima azienda a proporre modalità di comunicazione basate su XML. Però ha avuto la lungimiranza di non mantenere proprietarie le specifiche ma di renderle pubbliche. Questo ha portato ad un crescente interesse e

all'affermarsi dei primi standard che, ben presto, hanno visto numerose aziende intervenire in prima persona sia per estenderli che per supportarli.

1.9 PRESENZA DI TOOL

L'intero libro illustra alcuni dei numerosi tool adatti a creare e a fruire Web Services. La loro presenza è conseguenza di tutti i punti precedenti ma è l'unico vero motivo che riesce a catalizzare una quantità sempre crescente di sviluppatori; questo non fa altro che innescare una spirale di attività correlate che portano a definire nuove specifiche, aggregare nuove aziende, produrri altri tool e così via.

1.10 NON È TUTTO ORO...

C'è da prestare attenzione a non fare dei Web Services uno strumento di marketing fine a se stesso: oramai è di moda utilizzarli e sembra che qualsiasi applicazione che ne fa uso sia un'applicazione all'avanguardia. Ovviamente non è così: i Web Services sono un ottimo strumento dove l'interoperabilità è davvero un requisito essenziale. Per altri contesti, soprattutto in quelli in cui si ha un completo controllo sulle tecnologie adottabili nelle applicazioni coinvolte, i Web Services non sono una scelta ottimale: la loro "generalità" implica un volume di traffico notevole e non ottimizzato e un carico computazionale significativo sia per la codifica che la decodifica dei dati.

Proprio per questi svantaggi, tutte le piattaforme prevedono almeno un meccanismo di comunicazione remota proprietario: Java prevede RMI, Microsoft .NET prevede la tecnologia .NET Remoting ([Balena, 2004]) e così via.

Come accennato in precedenza si sta assistendo ad una esca-

PuntoInformatico Libri

lation di nuove specifiche che, per forza di cose, iniziano a diventare frammentarie (ciascuna affronta e risolve problemi molto specifici) e diverse aziende parteggiano per alcune soluzioni piuttosto che per altre mettendo a rischio il cuore stesso della tecnologia: l'interoperabilità.
Per fortuna questi usi particolari sono presenti solo in applicazioni di tipo Enterprise che, già di per sé, possiedono notevole complessità interna; questo libro darà solo un cenno ai diversi problemi "aperti" e alle soluzioni proposte.

BIBLIOGRAFIA

[AAVV, 2001] "*Sistemi Informativi, Volume V – Sistemi distribuiti*", di E. Ansuini, A. Lioy, A. Massari, E. Melis, M. Mezzalana, G. Raiss, G. Cantucci, C. Simonelli, FrancoAngeli 2001

[Balena, 2004] "*Programmare Microsoft Visual Basic .NET versione 2003*", F. Balena, Mondatori Informatica, 2004

[Birrell, Nelson, 1984] "*Implementing Remote Procedure Calls*", ACM Transactions on Computer Systems, febbraio 1984

[CE, 1998] "L'informazione nel settore pubblico: una risorsa fondamentale per l'Europa - Libro verde sull'informazione del settore pubblico nella società dell'informazione". Commissione Europea. COM(98)585, 1998

[OMG, 1991] "*The Common Object Request Broker: Architecture and Specification*", OMG 1991

[Voth et al., 1998] "*Distributed Application Development for Three-Tier Architectures: Microsoft on Windows DNA*", G. Voth e altri, IEEE Internet Computing, marzo 1998

[WSA, 2004] "*Web Services Architecture*", W3C Working Group Note 11 Febbraio 2004, *http://www.w3.org/TR/ws-arch/*

IL PRIMO WEB SERVICES IN… UN BATTIBALENO!

In questo capitolo si vedrà la semplicità di utilizzo di alcuni toolkit; in particolare si mostra l'uso di Axis (Java) e di Visual Studio .NET.La scelta di utilizzare un tool per Java e un ambiente di sviluppo della piattaforma .NET è giustificato dal fatto che queste sono, attualmente, le due piattaforme maggiormente utilizzate per realizzare Web Services e, nel corso del libro, saranno le due piattaforme a cui dedicheremo maggior approfondimento anche per illustrarne gli aspetti avanzati e di aderenza agli standard e alle problematiche di interoperabilità.

2.1 UN WEB SERVICE SEMPLICE SEMPLICE

Proviamo a realizzare, in Java, un semplicissimo Web Service e a interrogarlo utilizzando un linguaggio di programmazione a scelta.
Per esempio utilizziamo Visual Studio .Net con uno dei linguaggi messi a disposizione dalla piattaforma.Il servizio realizzato farà uso di Axis, uno dei tool che verranno approfonditi nei capitoli successivi. La scelta di realizzare ex novo un servizio Web è dettata dal fatto che anche in questo modo si può verificare quanto facile sia il compito (attenzione a non pensare che sia sempre così: esempi più complessi, sviluppati nel seguito, richiederanno conoscenze ben maggiori sia per quanto concerne le tecnologie coinvolte che per quanto riguarda il linguaggio Java). L'alternativa sarebbe stata quella di utilizzare un servizio reso disponibile gratuitamente e reperibile sul Web; per esempio si vedano i servizi esposti nel sito *www.xmethods.net*.

2.2 INSTALLARE AXIS, TOMCAT E IL JAVA DEVELOPER KIT

Prima di tutto è necessario preparare l'ambiente di sviluppo. Quest'ambiente è quello base per proseguire con gli esempi del resto del libro e pre-

PuntoInformatico | Libri

vede che debba essere installato il Java Developer Kit (JDK), un Servlet Container, per esempio Tomcat, e il framework Axis, specifico per lo sviluppo di Web Services; procediamo con ordine.Supponendo di lavorare con Windows, potete scaricare il file eseguibile per l'installazione di Tomcat dalla pagina *http://jakarta.apache.org/tomcat/*. L'ultima versione disponibile al momento di scrivere il libro è la 5.5.9.Eseguendo il file exe di cui si è fatto il download, si avrà a disposizione un wizard automatico. Si consiglia di installare la versione "Full", completa di esempi e documentazione **(Figura 2.2).** La versione full provvede anche a installarsi come servizio di Windows (per le versioni NT 4.0, 2000 ed Xp): in questo modo è possibile far partire il Servlet Container all'avvio del sistema operativo. Durante la configurazione, è importante ricordarsi la porta a cui il server risponde (quella di default è la 8080 ma è sempre possibile modificarla) e lo username/password dell'amministratore. Finita l'installazione si può verificare se Tomcat è attivo accedendo all'url *http://localhost:<porta>*.

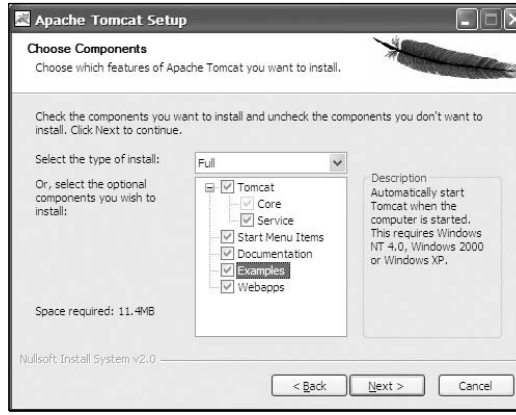

Figura 2.2: installazione, completa, di Tomcat.

In (**Figura 2.3)** ecco come dovrebbe apparire la pagina di benvenuto di default.

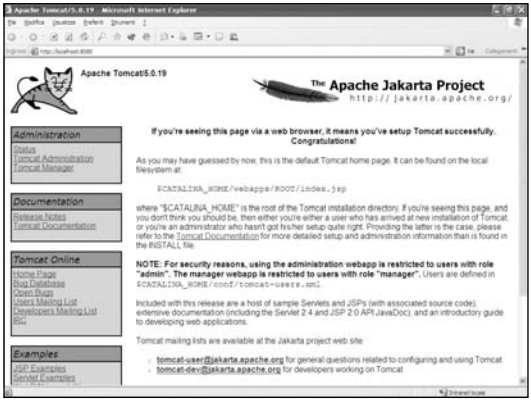

Figura 2.3: la pagina di benvenuto quando Tomcat è installato correttamente.

Ora non resta che installare la web application di Axis. Essa si trova nella directory ove è stato compattato il file sotto il path /webapps. Essa va copiata sotto la cartella /webapps di tomcat.

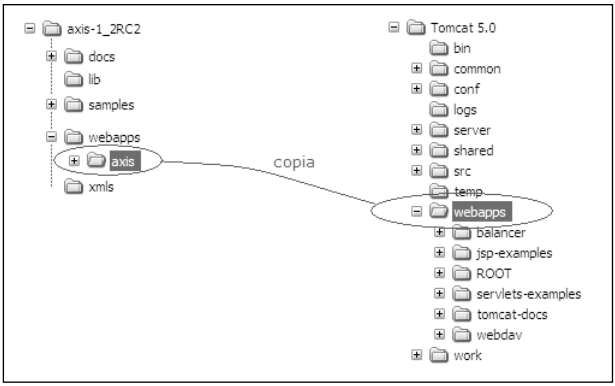

Figura 2.4: copiare la webapp di Axis.

Per verificare che l'applicazione si sia installata correttamente basta accedere alla pagina *http://localhost:<porta>/axis* e,

se viene visualizzata la pagina di Axis **(Figura 2.5)**, validare l'installazione seguendo il link "Validate".

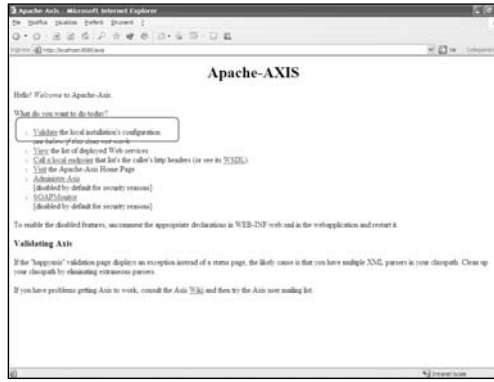

Figura 2.5: Axis risponde… ora si deve validare!

Sfortunatamente la validazione non va a buon fine. Infatti si può osservare che manca una componente tra quelle fondamentali.

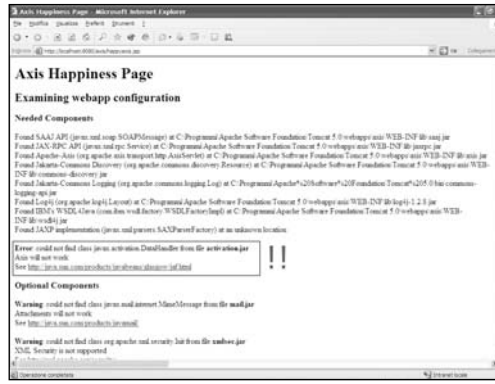

Figura 2.6: viene segnalato un problema che non permette ad Axis di funzionare correttamente.

L'errore segnala anche il file jar mancante. Non resta che scaricare tale file e copiarlo nella directory `/common/lib` dov'è installato Tomcat. Ora la pagina viene validata e abbiamo l'ambiente di sviluppo Java funzionante **(Figura 2.6)**!

2.3 IL PRIMO WEB SERVICE? UNA CLASSE JAVA

Costruiamo una classe Java minimale, contenuta nel file primoWS.java, con un solo metodo: `status()`. Facciamo rispondere a tale metodo il valore "Running":

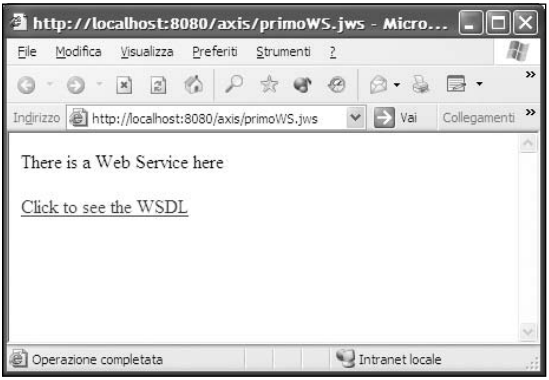

Figura 2.7: qui c'è un Web Services… e lo abbiamo appena creato!

```
public class primoWS {
  public String status(){
    return "Running";
  }
}
```

Questa è una classe Java valida. Ora la si copi con il nome `primoWS.jws` (si noti l'estensione) sotto la `$TOMCAT/webap-`

`ps/axis`. Ora, accedendo all'url *http://localhost:8080/axis/primoWS.jws* (al posto di 8080 si scriva la porta utilizzata, se diversa) appare una scritta che ci indica che c'è un Web Services a questa URL (**Figura 2.7**); cliccando sul link "Click to see the WSDL" apparirà un file; tale file è un file XML (l'url è *http://localhost:8080/axis/primoWS.jws?wsdl*; teniamola a mente perché la riutilizzeremo tra breve) .

2.4 OTTENERE IL... "CONTRATTO"

Per invocare un servizio Web dobbiamo conoscerne le caratteristiche (quali dati utilizza, eventuali parametri da passare, il loro tipo, l'ordine e così via). Potremmo pensare a queste caratteristiche come ad un'interfaccia da implementare. Però l'implementazione di interfacce è più consona in contesti di stretta dipendenza architetturale (stesso linguaggio, stessa piattaforma). Quindi, nel contesto dei Web Services, possiamo pensare che queste caratteristiche siano una specifica simile ad un contratto, ove tale contratto elenca le caratteristiche salienti da rispettare, ma lascia libertà nella sua implementazione.
Il contratto è scritto su un documento XML apposito (WSDL) ed è proprio quello che Axis ha creato in automatico dal Web Ser-

Figura 2.8: il WSDL del servizio Web (generato da Axis).

vice precedente (in pratica sono stati forniti i dettagli del servizio, sotto forma di una classe Java, e poi Axis si è preoccupato di generare il "contratto").
Ora si è pronti per far "consumare" il servizio ad un client. Si vedrà come realizzarlo utilizzando un secondo strumento di sviluppo: il Visual Studio .NET (la cui installazione è talmente facile che non verrà descritta!).

2.5 REALIZZARE IL CLIENT IN VB.NET

Qualsiasi applicazione può divenire un client di un Web Service, purché essa possa inviare e ricevere opportuni messaggi. Un client può essere sia un'applicazione stand alone (gli exe di Windows, tanto per intenderci), sia una pagina Web, un ulteriore Web Service e così via.
Per realizzare un qualsiasi programma in uno dei linguaggi della piattaforma .NET basterebbe possedere Microsoft .NET Framework (che, tra le altre cose, è gratuito). Però è opportuno avere a disposizione l'editor Visual Studio che agevola e semplifica enormemente lo sviluppo dei programmi.

Nota: benché il Visual Studio sia un prodotto commerciale, è possibile scaricare una versione di prova e valida 60 giorni (attivabile via Web) dalla pagina *http://msdn.microsoft.com/vstudio/*. Spesso, negli eventi Microsoft dedicati agli sviluppatori, vengono distribuiti CD-Rom che contengono Visual Studio (più, di solito, altro materiale relativo alla documentazione o a servizi extra).

Una volta installato il prodotto, si apra Visual Studio .NET e si crei un nuovo progetto. La scelta del linguaggio non è vincolante; per esempio si scelga di realizzare un progetto Visual Basic .NET di tipo "Applicazione per Windows" chiamato *Win-*

PuntoInformatico | Libri

dowsApplication_ClientPrimoWS. (se si ha dimestichezza con qualsiasi altro linguaggio .NET lo si utilizzi pure).
Appare una nuova form che va personalizzata; si inserisca un campo di testo e un pulsante, come da (**Figura 2.10)**.

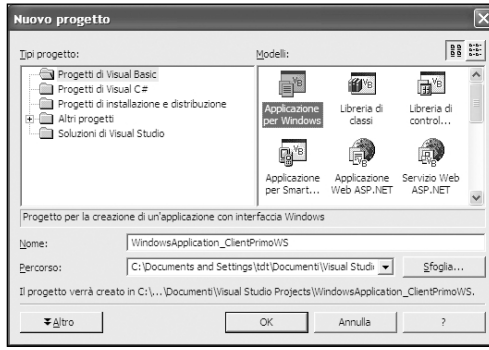

Figura 2.9: una nuova applicazione per Windows in Visual Studio.

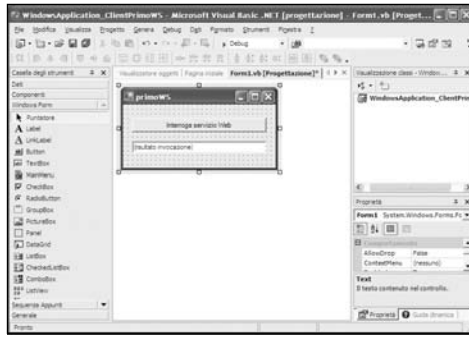

Figura 2.10: La form personalizzata.

Si vuole fare in modo che, quando l'utente preme sul pulsante, venga interrogato il Web Service realizzato in precedenza e accessibile alla URL di Tomcat; il risultato di tale invocazione (il messaggio del metodo "status") sarà visualizzato nel campo di te-

sto. Si vada su "Progetto > Aggiungi riferimento Web…": è possibile specificare una URL oppure eseguire una ricerca di Web Services. Nel caso di esempio l'unica azione possibile è specificare l'URL; in particolare va specificata l'url a cui risponde il WSDL(ovvero *http://localhost:8080/axis/primoWS.jws?wsdl*); premendo su "Vai" (freccia verde) verrà ricercato il Web Service e, se trovato, verrà mostrata la lista dei suoi metodi (uno, in questo caso, come mostrato in **Figura 2.11**).

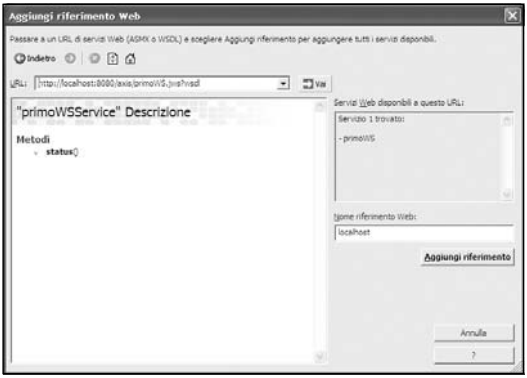

Figura 2.11: primoWS è stato riconosciuto!

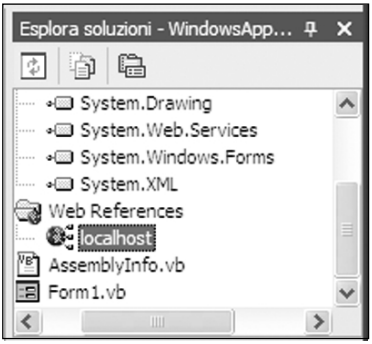

Figura 2.12: Aggiunto un riferimento al Web Service nel progetto.

Non resta che premere il pulsante "Aggiungi riferimento": in questo modo verrà aggiunto un riferimento al Web Service nel progetto (**figura 2.12**) che si concretizza nella generazione di una classe che espone metodi che hanno lo stesso nome del metodo del Web Service riferito.
Per visualizzare i dettagli della classe client generata, si vada sul *"Web References > Localhost"* e si clicchi con il pulsante destro: dal menu a tendina si scelga *"Visualizza nel Visualizzatore Oggetti"*. Ora viene mostrata la classe che si può selezionare e si possono verificare i metodi, che sono:

New()
BeginStatus(…)
EndStatus(…)
Status()

Il primo è il costruttore, gli altri tre permettono di invocare il metodo Status del servizio Web. In particolare `Status()` effettua una chiamata sincrona, ovvero l'esecuzione del programma si ferma in attesa del risultato del metodo, `BeginStatus(...)` inizia una chiamata asincrona che verrà terminata da `EndStatus(...)` (quest'ultimo metodo permetterà di reperire il risultato del metodo, sempre ché l'esecuzione remota del metodo sia terminata, solo nel momento che si ritiene più

Figura 2.13: Premendo il pulsante si invoca il Web Service e si scrive il risultato sulla casella di testo.

opportuno; nel frattempo è possibile continuare la normale esecuzione del programma). Non resta che invocare il servizio Web alla pressione del pulsante sulla form; dalla form eseguire un doppio click sul pulsante: si apre il metodo "Button1_Click", ovvero il metodo che gestisce l'evento Click sul pulsante "Button1" (per chi non conosce il VB.NET consiglio di far riferimento al libro [Balena, 2004]).
In questo metodo basterà inserire il seguente codice:

```
Dim servizioWeb As New localhost.primoWSService
TextBox1.Text() = servizioWeb.status()
```

La prima riga invoca il costruttore New()per creare una nuova istanza del client, la seconda invoca l'operazione status sul servizio Web e assegna il risultato alla casella di testo. Basterà mandare in esecuzione il progetto (premendo [F5]) e, sulla form, premendo il pulsante si avrà il risultato mostrato in Figura 2.13. Abbiamo appena creato un server che espone un servizio Web e un client che lo utilizza!

2.6 TUTTO QUI?

Come si è visto, la realizzazione di un client è un'operazione assai semplice. Ovviamente l'invocazione e il reperimento di un risultato sono assai facili: sarà poi necessario creare un'applicazione "attorno" che renda tale dato significativo. Quel che più conta è che, mentre l'implementazione di un Web Service può essere anche molto complessa, il client che lo utilizza può essere realizzato in maniera molto semplice come è stato appena descritto. Anche la realizzazione di un servizio Web, usando Axis, può sembrare piuttosto facile: lo è fintantoché non si hanno esigenze particolari (come, purtroppo, accade quasi sempre per progetti concreti!). È importante anche capire quali sono le tecnologie sottostanti i Web Services sia per capirne i possibili problemi sia per creare applicazioni sofisticate con caratteristiche fondamentali per ambienti di produzione, co-

PuntoInformatico | Libri

me interoperabilità con dati complessi, gestione della sicurezza e così via. Vediamo di introdurre le tecnologie di base dei servizi Web e, successivamente, di approfondire le problematiche citate...

BIBLIOGRAFIA

[AXIS, 2004] "*Programming Apache Axis*", AA.VV., O'Reilly, 2004

[Balena, 2004] "*Programmare Microsoft Visual Basic .NET versione 2003*", F. Balena, Mondadori Informatica, 2004

[Itani & Basha, 2002] "*AXIS Next Generation Java SOAP*", R. Irani, S. J. Basha, Wrox Press Ltd., 2002

[Tomcat, 2004] "*Professional Apache Tomcat 5*", AA.VV., Wrox ,2004

TECNOLOGIE SOTTOSTANTI I WEB SERVICES

3.1 INTERNET E LO STACK OSI

Internet è realizzato grazie ad un insieme di tecnologie standard, che permettono di far comunicare applicazioni diverse. La complessità delle diverse architetture ha portato a separare, logicamente, diversi strati software, in base ai "ruoli" che ricoprono durante la comunicazione. Infatti, quando si parla di applicazioni che "comunicano" con altre applicazioni in un'architettura distribuita, si è soliti utilizzare lo stack OSI, ovvero utilizzare un modello che descrive diversi strati software che realizzano una comunicazione. In (**Figura 3.1)** è mostrato tale stack (nell'alto le tecnologie e gli standard di più alto livello di astrazione).Questo schema concettuale ci aiuterà anche a collocare opportunamente gli standard introdotti per i Web Services.

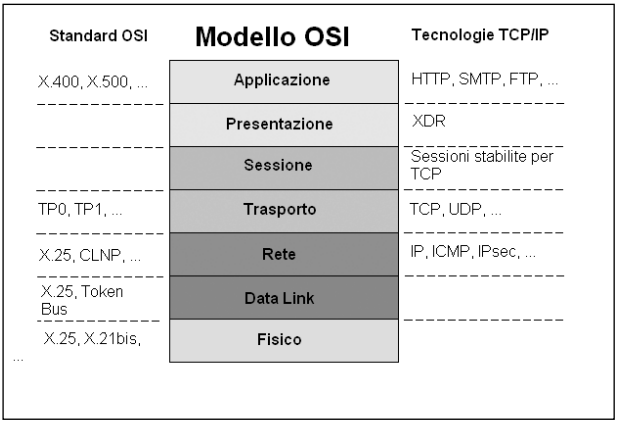

Figura 3.1: le tecnologie coinvolte e lo stack OSI

XML: EXTENSIBLE MARKUP LANGUAGE

3.2 XML E TECNOLOGIE CONNESSE

L'XML (eXtensible Markup Language) nasce come linguaggio di rappresentazione indipendente dalla piattaforma e da qualsiasi implementazione specifica; il suo "antenato" è il linguaggio SGML. Rispetto ad SGML è più semplice e, grazie a queste semplificazioni, è molto più facile realizzare parser per XML che per SGML. Questa semplicità è dovuta anche ad una maggior "rigidezza" nella definizione di documenti XML validi, ma non significa certo meno flessibilità nei contesti applicativi.
L'XML è leggibile anche da persone umane, nel senso che viene usata una rappresentazione con caratteri alfanumerici, ma viene utilizzato soprattutto per scambiare dati tra applicazioni. Per poter essere interpretato correttamente ogni documento XML deve seguire delle precise regole formali (regole sintattiche). La semantica di un file XML, invece, è dipendente dal contesto. È un linguaggio a marcatori (tag), dove ogni marcatore può essere singolo o a coppie. Nel primo caso la sua rappresentazione è <marcatore />, nel secondo <marcatore>dati</marcatore> (si noti la similitudine rispetto ai tag HTML). Un marcatore viene detto elemento XML.
Si possono avere elementi annidati:

```
<m1><m2>valore</m2></m1>
```

In questo caso si può dire che m2 è un sotto-elemento di m1; in ogni caso è necessario che sia rispettata una gerarchia, ovvero è possibile che più elementi siano contenuti come delle "matriosche" ma non sono ammesse sovrapposizioni; pertanto

```
<m1><m2></m1></m2>
```

non è un documento XML corretto.
Un documento XML, inoltre, deve fornire un marcatore iniziale particolare, che ne definisce la natura e la versione dello standard XML a cui aderisce: <?xml version="1.0" ?>
Ecco un esempio di documento XML ben formato, dove per ben formato si intende che risponde alle regole sintattiche dello standard:

```
<?xml version="1.0">
<prodotto>
  <id> 123</id>
  <descrizione>Mouse</descrizione>
  <prezzo>12</prezzo>
  <valuta>Euro</valuta>
</prodotto>
```

Chiunque, leggendo il file XML, può intuirne il significato dell'oggetto (o degli oggetti) rappresentato. Ma questa intuizione non è utile né significativa per un programma. Neppure a livello intuitivo sapremmo dire se i dati rappresentati siano tutti validi né se siano rappresentati tutti i dati essenziali affinché delle applicazioni li possano interpretare correttamente.
Per definire quali documenti XML abbiano queste caratteristiche è necessario definire un secondo tipo di documenti: degli schemi XML (XML schema) o dei documenti di dichiarazione di tipo (Document Type Declaration o DTD), che non sono documenti XML che rappresentano dati, ma documenti XML che descrivono quali documenti XML sono validi secondo le regole espresse nello schema (in pratica dato uno schema è possibile sapere se un documento soddisfa la sua definizione oppure no; in termini formali descrive la grammatica secondo cui i documenti XML validi devono essere scritti).
Prima di descrivere questi oggetti vediamo altre due definizioni: XML Namespaces e attributi XML.

PuntoInformatico Libri

3.3 ATTRIBUTI XML

Un attributo XML è contenuto in un elemento XML; l'elemento che può contenere degli attributi è l'elemento formato da un unico marcatore oppure quello iniziale (non quello di chiusura!):

```
<marcatore attributo="valore" />
<marcatore attributo="valore"> …</marcatore>
```

Ogni elemento può avere al più un attributo con lo stesso nome. Pertanto, mentre il seguente frammento XML è corretto:

```
<prodotto id="123" tipo="alimentare" prezzo="2" valuta="Euro" />
```

Questo non lo è:

```
<prodotto id ="1" tipo="alimentare" tipo="altro" prezzo="2"
                                                valuta="Euro" />
```

Invece si potrebbe scrivere questo documento XML dove, al posto degli attributi, si usano sotto-elementi XML:

```
<prodotto>
 <tipo>alimentare</tipo>
 <tipo>altro</tipo>
<!— altro —>
</prodotto>
```

Questo sarebbe un documento XML corretto anche se esistono più sotto-elementi dello stesso tipo.

3.4 ATTRIBUTI O ELEMENTI?

Quando usare attributi e quando utilizzare gli elementi? Non esiste una regola precisa. Però si potrebbe pensare di utilizzare gli at-

tributi quando il loro valore è legato in maniera stretta all'elemento a cui si potrebbero riferire. Per esempio se utilizziamo "id" per identificare uno ed un solo prodotto, è plausibile che esso sia un attributo. Allo stesso modo se usiamo un prezzo, è verosimile che la valuta sia un attributo del prezzo (anche perché se utilizzassimo due prezzi, espressi in valute diverse, dovremmo specificare due volte l'elemento prezzo e così pure l'elemento valuta: un modo semplice per "legare" i due è utilizzarne uno come attributo). Pertanto potremmo pensare di "esprimere" un prodotto in questo modo:

```
<?xml version="1.0">
<prodotto id="123">
  <prezzo valuta="Euro">12</prezzo>
  <descrizione>Mouse</descrizione>
  <quantita>1</quantita>
</prodotto>
```

3.5 XML NAMESPACE

Un namespace definisce un insieme di nomi unico in uno specifico contesto. In concreto si usano i namespace per evitare conflitti tra nomi (riferiti ad attributi o elementi). Infatti è verosimile che chiunque scriva un documento XML che descrive dei prodotti commerciali possa usare uno o più nomi utilizzati da qualcun altro nei suoi documenti XML (però mentre il marcatore prodotto in un contesto ha un significato, lo stesso marcatore in un altro contesto avrà sicuramente un significato diverso, dove per significato si intende l'insieme degli elementi e degli attributi ammessi). Per evitare confusioni è necessario dichiarare uno spazio univoco dei nomi che permette di disambiguare tutti i nomi utilizzati. Ogni namespace usa un URN (Unifom Resource Name) della forma:

```
xmlns:NameSpaceID="NameSpaceSpecificString"
```

PuntoInformatico | Libri

Per esempio si può utilizzare il namespace `xmlns:prodotti=http://www.ioprogrammo.it/prodotti/` e `xmlns:fornitori=http://www.ioprogrammo.it/fornitori/;` se si usasse un elemento "nome" in entrambi, con prodotti:nome si è certi di riferirsi ad un elemento definito nel primo namespace, con fornitori:nome ad un elemento del secondo.

3.6 XML SCHEMA E DTD

Gli XML schema e i DTD sono entrambi utilizzati per definire gli elementi di un documento XML; però gli schemi XML permettono anche di specificare dichiarazioni di tipo, vincolando i range dei valori assumibili. Nel contesto dei Web Services è opportuno approfondire solo gli XML Schema in quanto i DTD non sono utilizzabili per i messaggi SOAP a partire dalla specifica 1.2. Un XML Schema usa un linguaggio chiamato XML Schema Definition language (abbreviato in XSD) con proprie regole di composizione dei documenti. Esso è, fondamentalmente, un insieme di tipi predefiniti e un modo per definirne di nuovi.

3.7 XML SCHEMA PER LA DEFINIZIONE PRECISA DEI DATI

Quelli predefiniti sono riportati in tabella. Si noti che sono tutti dei tipi semplici, chiamati anche scalari, e non strutture dati complesse. Queste ultime sono "costruibili" a partire dai tipi base e usando alcune regole, che ora andiamo a definire.
Tali tipi di base sono chiamati tipi semplici predefiniti (**Tabella 3.2 e 3.3**).

Nota: I tipi contrassegnati da (*) possono essere rappresentati in diversi formati: per esempio 1000 e 1.0E3 sono equivalenti per rappresentare una quantità di "mille"

Tipo (predefinito)	Possibili valori
string	Una qualsiasi stringa di valori
normalizedString	Come il tipo string, ma eventuali caratteri di nuova linea, tabulazione e ritorno a capo vengono convertiti
token	Come il tipo normalizedStringa ma caratteri vuoti (spazi) adiacenti sono convertiti in un unico carattere di spazio. Inoltre se esistono spazi all'inizio o alla fine della stringa, questi vengono eliminati (operazione di trim).
boolean	Valori di verità(booleani) rappresentati da true,false, 0 e 1
base64Binary	Dati binari rappresentati in base64
hexBinary	Dati binari in formato esadecimale
integer (*)	Valori interi (sia positivi che negativi)
positiveInteger (*)	Solo interi positivi (lo 0 non è considerato positivo!)
negativeInteger (*)	Solo valori interi negativi (lo 0 non è considerato negativo!)
nonPositiveInteger (*)	Valori interi non positivi (ovvero negativi e lo 0)
nonNegativeInteger (*)	Valori interi non negativi (ovvero positivi e lo 0)
long (*)	Interi lunghi (valori da -9223372036854775808 a 9223372036854775807)
unsignedLong (*)	Interi lunghi senza segno (valori da 0 a 18446744073709551615)
int (*)	Interi (valori da -2147483648 a 2147483647)
unsignedInt (*)	Interi senza segno (valori da 0 a 4294967295)
short (*)	Interi corti (valori da -32768 a 32767)
unsignedShort (*)	Interi corti senza segno (valori da 0 a 65535)
byte (*)	Valori interi da -128 a 127
unsignedByte (*)	Valori interi senza segno da 0 a 255
decimal (*)	Valori decimali
float (*)	Valori in virgola mobile a precisione singola (32 bit), compresi meno infinito (-INF) e infinito (INF) e valore NaN (Not a Number)
double (*)	Valori in virgola mobile a precisione doppia (64 bit), compresi meno infinito (-INF) e infinito (INF) e valore NaN (Not a Number)

PuntoInformatico Libri

Tipo (predefinito)	Possibili valori
Duration	Valori temporali (relativi)
dateTime (*)	Valori temporale (date e orari)
date (*)	Valori temporali (solo date)
time (*)	Valori temporali (solo orari)
gYear (*)	Valore intero rappresentante un anno
gYearMonth (*	Valore rappresentante anno e mese (2005-05)
gMonth (*)	Rappresenta un mese (—05)
gMonthDay (*)	Rappresenta mese e giorno (—05-15)
gDay (*)	appresenta un giorno (—15

Tabella 3.2: I tipi semplici predefiniti di un XML Schema

Le date sono sempre rappresentate nella forma AAAA-MM-GG, ovvero anno, mese, giorno (per chi conosce Java è il formato delle date secondo lo standard JDBC). Inoltre sono rappresentabili, sempre come tipi semplici, valori definiti nella specifica XML 1.0 (**Tabella 3.3**).

Al fine di rendere compatibili DTD di XML 1.0 e XML Schema, tutti i tipi definiti nella (**Tabella 3.3)** successivi a language dovrebbero essere utilizzati unicamente negli attributi. Come si può notare dallo schema di (**Figura 3.4)**, esiste una gerarchia tra i tipi di dato pre-

Tipo (predefinito)	Valori
Name	Name Type
Qname	Namespace Qname
NCName	Namespace NCName, ovvero è un QName senza il prefisso
anyURI	Qualsiasi Unified Resource Identifier
Language	Lingua e nazione (secondo lo standard ISO 639, si veda http://www.w3.org/WAI/ER/IG/ert/iso639.htm); per esempio en-GB (linguaggio inglese, nazione Gran Bretagna, o it-IT linguaggio italiano e nazione Italia).
ID	ID attribute type
IDREF	IDREF attribute type

Tipo (predefinito)	Possibili valori
IDREFS	IDREFS attribute type
ENTITY	ENTITY attribute type
ENTITIES	ENTITIES attribute type
NOTATION	NOTATION attribute type
NMTOKEN	NMTOKEN attribute type
NMTOKENS	NMTOKENS attribute type

Tabella 3.3: i tipi semplici riferiti ad XML 1.0 predefiniti di un XML Schema

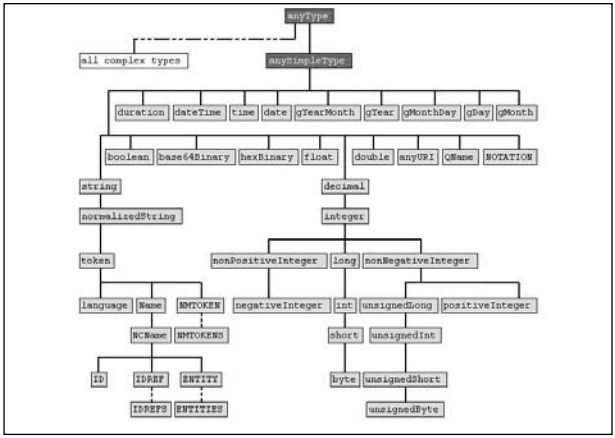

Figura 3.4: La gerarchia dei tipi predefiniti (figura tratta da [W3CSchema2, 2004])

definiti.

3.8 COSTRUIRE UN XML SCHEMA

Uno schema XML inizia con l'elemento "schema". Si è soliti utilizzare anche il prefisso xsd, anche se questo non è vincolante e si può utilizzare un qualunque prefisso, e si definisce il namespace come segue:

```
<xsd:schemamlns:xsd="http://www.w3.org/2001/XMLSchema">
```


PuntoInformatico Libri

```
</xsd:schema>
```

All'interno è possibile inserire altri sottoelementi; i più significativi sono element, simpleType e complexType.Un simpleType può essere uno dei tipi predefiniti ma a cui si applicano ulteriori vincoli, specifici per lo schema in uso; ecco, per esempio, come definire un nuovo tipo, ordinabile, che assume valori interi, ma i cui valori ammissibili vanno da 1 a 999:

```
<xsd:simpleType name="ordinabile">
 <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="1"/>
  <xsd:maxInclusive value="999"/>
 </xsd:restriction>
</xsd:simpleType>
```

È possibile creare delle enumerazioni (utili quando i valori ammissibili sono univocamente determinati a priori):

```
<xsd:simpleType name="possibiliValute">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="Euro"/>
  <xsd:enumeration value="Dollaro"/>
  <xsd:enumeration value="Yen"/>
  <!— altre ... —>
 </xsd:restriction>
</xsd:simpleType>
```

A questo punto è possibile definire un tipo di dato (complesso) che è un prezzo espresso secondo una determinata valuta. È possibile definire tipi complessi utilizzando l'elemento complexType. Al suo interno è possibile dichiarare elementi con "element", attributi con "attribute".Ogni elemento ha un nome (name) e un tipo (type) che è uno dei tipi complessi definiti dall'utente o uno dei tipi

predefiniti (**Tabella 3.2 e 3.3**).

```
<xsd:complexType name="prezzoInternazionale">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="valuta" type="possibiliValute" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Potremmo definire un prodotto con il seguente schema:

```
<xsd:complexType name="prodotto" >
 <xsd:sequence>
  <xsd:element name="tipo" type="xsd:NMTOKEN" maxOccurs="3" />
  <xsd:element name="prezzo" type="prezzoInternazionale" />
  <xsd:element name="descrizione"  type="xsd:string" minOccurs="0" />
  <xsd:element name="quantita" type="ordinabile" />
 </xsd:sequence>
 <xsd:attribute name="id" type="xsd:decimal" use="required" />
</xsd:complexType>
```

Si noti l'uso degli attributi maxOccurs, che specificano il numero massimo di elementi di quel tipo, minOccurs indica il numero minimo, use specifica l'uso (può essere required se è obbligatorio specificarlo, optional se è opzionale e prohibited se è vietato usarlo; quando è specificato optional si può usare anche l'attributo default per indicare un valore di default). Con quanto visto abbiamo esaurito la definizione dell'elemento "prodotto". Se si vogliono inserire dei commenti e far sì che essi siano parte del documento si può ricorrere all'elemento annotation:

```
<xsd:annotation>
  <xsd:documentation>Descrizione</xsd:documentation>
```

PuntoInformatico Libri

```
</xsd:annotation>
```

Per ulteriori informazioni sugli XML Schema si veda [W3CSchema, 2004].

3.9 SOAP, WSDL E UDDI

Per far colloquiare due o più applicazioni eterogenee (per tecnologia) e senza tipi di dato comuni, è necessario gestire un qualsiasi protocollo di comunicazione non legato ad uno specifico ambiente. Come tutti i protocolli, esso deve essere non ambiguo e quanto più generale: ecco che è nato il protocollo di comunicazione "a messaggi", dove ogni messaggio è codificato secondo uno standard, chiamato SOAP; accanto ad esso è stato definito un formalismo, un metalinguaggio (WSDL), che ne descrive i possibili tipi di messaggi che un servizio "comprende" e, infine, è stato studiato un registro ove chi vuol rendere pubblici i propri servizi Web li registra e, a seguito di tale registrazione, un client può reperirlo attraverso un meccanismo di ricerca (UDDI). Vediamo nel dettaglio le tecnologie appena citate.

3.10 SOAP

SOAP è giunto alla versione 1.2 dello standard. L'evoluzione rispetto alle versioni precedenti non è sempre stata indolore: infatti alcuni utilizzi delle specifiche precedenti tutt'oggi "inquinano" l'utilizzo pulito della nuova specifica. Ma vediamo, in concreto, cosa definisce tale specifica…

3.11 I MESSAGGI

Un servizio Web viene invocato attraverso dei messaggi che sono scritti secondo il formato SOAP, così pure le risposte (se presenti) sono codificate secondo tale formalismo. SOAP in origine era l'acroni-

mo di *Simple Object Access Protocol*, ma a seguito all'evolversi del formalismo tale acronimo ha perso il suo significato originario. Infatti, ad oggi, i messaggi SOAP, possono anche non essere invocazioni remote su oggetti. Ma com'è formato, in concreto, un messaggio SOAP? Esso è espresso da un documento XML aventi le seguenti caratteristiche: ogni messaggio è racchiuso in un elemento chiamato envelope (involucro) che al suo interno deve contenere un sottoelemento body. Opzionalmente può contenere anche una intestazione, contenuta nel sotto-elemento header (**Figura 3.5**).
All'interno di envelope vanno inseriti anche i namespace utilizzati

3.12 IL PROBLEMA DEL TRASPORTO (HTTP, SMTP, ...)

SOAP è un protocollo che si colloca a livello application-layer (si riveda lo stack OSI di inizio Capitolo) e come tale risulta indipendente dal protocollo di comunicazione usato. Eppure in origine (specifica SOAP v1.0) SOAP era legato al protocollo di trasporto Http. L'evoluzione (specifica SOAP 1.1 e 1.2) ha portato ad abbandonare

PuntoInformatico | Libri

```
<soap:Envelope>

    <soap:Header>
      <!-- intestazione -->
    </soap:Header>

    <soap:Body>
      <!-- contenuto -->
    </soap:Body>

</soap:Envelope>
```

Figura 3.5: La struttura di un messaggio SOAP

tale connubio e a rendere possibile l'uso di uno qualunque dei protocolli di trasporto esistenti. Benché esista questa possibilità, e difficilmente si possono fare assunzioni contrarie, negli esempi del libro faremo uso del protocollo Http.

3.13 IL PROBLEMA DELLA SICUREZZA (HTTPS; WS-SECURITY, ...)

Quando si parla di sicurezza è necessario prevedere scelte architetturali che riguardano l'autenticazione e l'autorizzazione degli utenti nonché il modo in cui tali informazioni debbano essere presentate. Infatti le chiamate SOAP avvengono tra applicazioni ma, di solito, è necessario prevedere una qualche autenticazione dell'utente che utilizza l'applicazione che esegue la chiamata SOAP. Questa autenticazione può avvenire nelle forme usuali (form di inserimento dati, certificati X.509 esposti all'applicazione, o altre forme più avanzate). Una volta autenticato l'utente è possibile associare i suoi dati a livello di trasporto (è il caso di Http con username/password o Https con certificati X.509). Questa scelta però è valida solo se il trasporto è sempre realizzato con la stessa tecnologia. Cosa accade se invece il trasporto avviene per una parte in Http, una in SMTP e una via FTP? Fare assunzioni su quale trasporto debba essere usato limita l'ambito di applicazione e non è detto che questo vincolo sia sempre rispettato. Pertanto si cerca di trasferire la sicurezza direttamente a livello di messaggio SOAP o attraverso opportune estensioni della specifica.

3.14 SCELTE ARCHITETTURALI

Un'altra scelta importante, in termini di sicurezza, è quando e in che modo eseguire i controlli sui messaggi SOAP. Questa scelta è di tipo architetturale e prevede, essenzial-

mente, tre casi:

il controllo avviene a livello di gateway. Questo gateway può risiedere anche su un nodo diverso rispetto al nodo che espone il servizio. Il controllo avviene da un "interceptor", ovvero un modulo software all'interno del nodo che espone il servizio, ma questo modulo è centralizzato e può essere valido per un insieme di servizi. Il controllo viene implementato all'interno del servizio stesso.

In (**Figura 3.6)** sono schematizzate le tre scelte archietturali appena de-

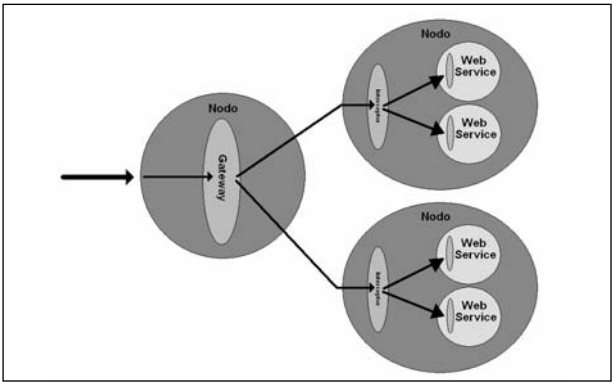

Figura 3.6: Le scelte architetturali per affrontare il problema della sicurezza

scritte.

Le scelte sono state elencate in ordine di "astrazione" decrescente, con conseguente maggior isolamento della logica del controllo per il gataway, un isolamento parziale per l'interceptor e nessun isolamento, se non dato dalla scrittura di funzioni e librerie, come accade nel terzo caso.

Minor astrazione, di solito, implica maggior controllo rispetto alle caratteristiche volute ma anche minor riuso delle funzionalità e necessità di manutenzione integrata al cambiamento delle specifiche di sicurezza nonché rispetto a modifiche delle funzionalità esistenti.

PuntoInformatico Libri

Non esiste, in assoluto, una scelta migliore di un'altra ma è importante riuscire a prendere il prima possibile una decisione su quale politica adottare (questo vale per tutte le scelte, ma in maniera preponderante per quelle architetturali).
Nel Capitolo 10 analizzeremo alcune proposte e standard accettati per rendere sicuri i servizi Web.

3.15 WSDL

Per descrivere i Web Services esiste un linguaggio formale, chiamato Web Services Definition Language (WSDL).
Esso definisce la grammatica che devono soddisfare i messaggi validi per un particolare Web Service. È un linguaggio formale ad alto livello, non dipendente dalla specifica implementazione del Web Service che descrive.Ogni file WSDL ha l'elemento principale "definitions", che contiene il nome del Web Service e dichiara tutti gli spazi dei nomi (namespace) XML utilizzati al suo interno:

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
  targetNamespace="your namespace here"
  xmlns:tns="your namespace here"
  xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap">
    <!— documento —>
</wsdl:definitions>
```

Alcuni esempi di namespace comunemente utilizzati sono sintetizzati in (**Tabella 3.7)**. Al suo interno ci sono diverse sezioni (schematizzate in **Figura 3.8**):
Types: è un contenitore di tipi di dato definiti attraverso un opportuno meccanismo di specifica (di solito XSD);
Message: una definizione astratta dei dati che sono scambiati nella forma di messaggi; la definizione comprende la specifica dei tipi coinvolti nella comunicazione;

Operation: le azioni esposte dal servizio (definizione astratta);
Port Type: un insieme di operazioni supportate da uno o più endpoint (insieme astratto);
Binding: un protocollo e una specifica sui formati dei dati concreti per un particolare Port Type;
Port: un endpoint (singolo) definito come combinazione di binding e indirizzi di rete;

Abbreviazione	Namespace	Significato
wsdl	http://schemas.xmlsoap.org/wsdl	default namespace
soapbind, wsdlsoap, o soap	http://schemas.xmlsoap.org/wsdl/soap	Gli elementi che specificano il binding per i messaggi SOAP
xsd	http://www.w3.org/2001/XMLSchema	Definizioni presenti in un XML Schema

Tabella 3.7: I namespace comunemente utilizzati all'interno di un WSDL

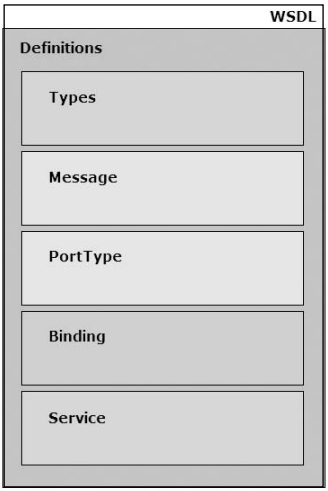

Figura 3.8: Le sezioni che compongono un documento WSDL

PuntoInformatico Libri

Service: una collezione di endpoint correlati.
Si noti che qualche elemento viene indicato come astratto o come concreto: la differenza è che i primi sono descrizioni di qualcosa che specifica il comportamento, i secondi specificano un'entità che ha un'implementazione ed è riferita ad un oggetto reale, concreto per l'appunto (è un po' quello che accade con il concetto di classe e oggetto nella programmazione ad oggetti: la classe specifica le proprietà, ma sono gli oggetti le entità con cui è possibile lavorare concretamente).

3.16 TYPES

All'interno di questa sezione vengono specificati i tipi e gli elementi, definiti da un XML Schema:

```
<wsdl:types>
  <xs:schema targetNamespace="namespace proprietario"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   <!— definizione di tipi ed elementi —>
  </schema>
</wsdl:types>
```

In questa sezione sono descritti tutti i tipi di dato utilizzati nel resto del documento. Quando i tipi di dato utilizzati sono quelli definiti dalle specifiche del XML Schema (si veda **Tabelle 3.2 e 3.3**) non è necessario specificarli; nel caso non esistano altri tipi di dati al di fuori di essi, questa sezione può essere omessa.

3.17 MESSAGE

Descrive diversi messaggi, ciascuno dei quali rappresenta un messaggio di tipo request o di tipo response (ovvero in ingresso ad una

operazione o come suo risultato):

```
<wsdl:message name="qualche op di request ">
  <!— parte(i) —>
</wsdl:message>
<wsdl:message name="qualche op di response">
   <!— parte(i) —>
</wsdl:message>
```

Si noti che viene definito il nome del messaggio, che può contenere una o più parti (ciascuna delle quali si riferisce a parametri del messaggio o a valori di ritorno).

3.18 PORTTYPE

Definisce le operazioni presenti nel servizio; tali definizioni avvengono attraverso i messaggi che compongono le operazioni:

```
<wsdl:portType name="your type name">
  <!— operazioni definite dai messaggi che le compongono  —>
</wsdl:portType>
```

Se c'è solo un messaggio di tipo request, si hanno comunicazioni ad una sola via (senza risultato); viceversa se esistono messaggi di request e messaggi di response, l'operazione è a due vie. È anche possibile definire operazioni con soli messaggi di response, anche se le operazioni più comuni sono le precedenti.

3.19 BINDING

Contiene lo stile dei messaggi, l'uso e il tipo di trasporto:

```
<wsdl:binding name="bindingName" type="tns:portTypeName">
  <!— stile, uso, trasporto delle operazioni —>
```

PuntoInformatico Libri

```
</wsdl:binding>
```

Queste informazioni avranno un notevole impatto sul modo di costruire i messaggi e meritano una riflessione specifica, che sarà fatta un po' più avanti.

3.20 SERVICE

Infine c'è la sezione *service*:

```
<wsdl:service>
  <!— definizione di port, usando binding e URL —>
</wsdl:service>
```

Quest'ultima parte si riferisce all'implementazione specifica del servizio, compreso di indirizzo di invocazione. Nel caso che il servizio SOAP sia reso disponibile su protocollo http, viene specificata la URL a cui il servizio risponde. La locazione fisica dove un servizio Web risponde è chiamata endpoint del servizio.

3.21 I FORMATI DEL MESSAGGIO

Si è detto che, all'interno della sezione binding, si specifica anche lo stile dei messaggi. Body può contenere due tipi di formati dei messaggi: document o rpc. Quest'ultimo si chiama così in quanto è quello usato per le invocazioni Remote Procedure Call e prevede che il Body contenga un elemento il cui nome è il nome della procedura remota da invocare. Invece se un messaggio ha il formato Document, allora il body contiene uno o più sotto-elementi, chiamati parti, ciascuna delle quali contiene documenti generici (ma il cui contenuto verrà formalizzato da un altro documento: il WSDL che descriveremo nel seguito). Benché lo stile RPC sia più "capibile" leggendo il Body del messaggio SOAP, lo stile document risulta più appropriato perché ga-

rantisce (almeno in teoria!) una miglior interoperabilità in quanto il WSDL che descrive il messaggio permette un controllo maggiore sulla costruzione dei messaggi ritenuti validi.

3.22 L'USO: ENCODED O LITERAL

L'uso dei messaggi specifica come i dati ivi contenuti debbano essere serializzati. Esistono due usi:

encoded (detto anche SOAP Encoded)
literal

Il primo è una specifica di SOAP 1.1 (oramai obsoleta con la specifica 1.2) che indica come certi oggetti, strutture ed array (ma anche grafi rappresentati strutture di oggetti complesse) debbano essere serializzate. Il secondo uso, literal, demanda ad un XML Schema il compito di definire le regole per la serializzazione.

3.23 UDDI

L'ultima tecnologia considerata è la Universal Description and Discovery Interface (UDDI). Già dal nome si capisce quali siano i servizi che offre un formalismo di descrizione (universale) di servizi; un'interfaccia di pubblicazione di nuovi servizi; un'interfaccia per il reperimento dei servizi pubblicati. Quando si crea un servizio Web pubblico è opportuno registrarlo in una o più directory UDDI per renderlo reperibile anche a chi non ne conosce l'endpoint. In pratica è come rendere un sito Web reperibile da un motore di ricerca al fine di farlo trovare anche da quegli utenti che ricercano determinate caratteristiche e non sono a conoscenza dell'URL del sito. Talvolta il client viene chiamato anche service consumer, il registro UDDI service registry e il fornitore del servizio Web service provider. UDDI diviene fondamentale in architetture di tipo SOA, mentre può essere

PuntoInformatico | Libri

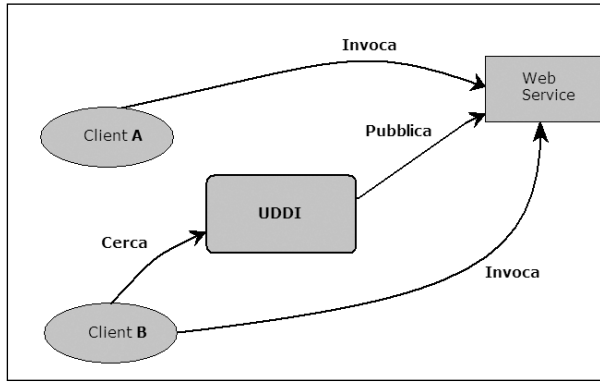

Figura 3.9: Un client può invocare un servizio Web conoscendone l'endpoint (client A) oppure a seguito di una ricerca su un registro UDDI (Client B)

di secondaria importanza in altre architetture meno "aperte".

3.24 CONCLUSIONI

Gli standard sottostanti i Web Services sono nati (ed evoluti) con una costante attenzione alla generalità degli stessi (si è evitato di vincolare il loro uso a dettagli implementativi).
Purtroppo il fatto stesso di avere a disposizione più versioni degli standard, nonché più modi di utilizzarne alcune caratteristiche, porta ad un problema di compatibilità tra tool diversi (problema non indifferente, visto che i Web Services nascono e si evolvono avendo tra gli obiettivi principali proprio l'indipendenza dal linguaggio o dalla piattaforma!).
Si potrebbe obiettare che questi sono problemi dei singoli tool, non della specifica: in parte questo è vero, ma è vero anche che la specifica va utilizzata con cognizione di causa.Valgono solo l'esperienza e un processo di continuo miglioramento basato su propri errori? No, per fortuna!
Esiste un insieme di specifiche che aiutano a trovare modalità d'uso dei formalismi il più possibile interoperabili. Questo in-

sieme di specifiche prende il nome di WS-I Basic Profile…

BIBLIOGRAFIA

[AAVV, 2004] *"Building Web Services with Java : Making Sense of XML, SOAP, WSDL, and UDDI"*, AA.VV., 2nd Edition, SAMS, 2004

[XMLNames, 1999] "*Namespaces in XML*", T. Bray, D. Hollander, A. Layman, http://www.w3.org/TR/xmlschema-0/

[W3CSchema, 2004] "*XML Schema Part 0: Primer Second Edition*", D. C. Fallside, P. Walmsley http://www.w3.org/TR/xml-schema-0/

[W3CSchema1, 2004] "*XML Schema Part 1: Structures Second Edition*", AA.VV, http://www.w3.org/TR/xmlschema-1/

[W3CSchema2, 2004] "*XML Schema Part 2: Datatypes Second Edition*", AA. VV., http://www.w3.org/TR/xmlschema-2/

[W3CSOAP, 2003] "*SOAP Version 1.2 Part 0: Primer*", N. Mitra, http://www.w3.org/TR/soap12-part0/

[WSDL, 2001] "*Web Services Description Language (WSDL) 1.1*", W3C Note, http://www.w3.org/TR/wsdl

PuntoInformatico | Libri

COLLABORARE? WS-I BASIC PROFILE!

WS-I (Web Services Interoperability) Basic Profile è una proposta della Interoperability Organization (sito di riferimento *http://www.ws-i.org*). L'organizzazione ha avuto origine nel 2002 grazie ad un gruppo di aziende, prime fra tutte l'IBM e la Microsoft, preoccupate del problema della cooperazione e della compatibilità fra i Web Services sviluppati con tool diversi (nel tempo si sono aggiunte praticamente tutte le maggiori aziende interessate al settore, quali Sun Microsystems, BEA, Oracle e molte altre). La specifica riguarda come le tecnologie legate ai Web Services debbano essere utilizzate. In pratica non aggiunge nulla di nuovo agli standard esistenti, ma fornisce indicazioni sul modo di utilizzare ciascuno degli standard coinvolti. Tali indicazioni sono sia documenti (specifiche) sia tool che permettono l'analisi dei manufatti (messaggi, descrittori e così via) al fine di certificarne l'aderenza alle specifiche; questa aderenza garantisce la compatibilità tra tutti i servizi Web che hanno superato la certificazione.L'esigenza di un simile insieme di specifiche nasce dal fatto che gli standard (primo fra tutti SOAP) si sono evoluti nel tempo: da un'esigenza iniziale si è passati all'aggiunta di nuove caratteristiche e nuove possibilità; purtroppo queste aggiunte non sempre sono "ortogonali" alle specifiche esistenti, permettendo di scrivere lo stesso tipo di messaggi/descrittori in maniera diversa. Tutti queste modalità sono lecite per quanto concerne lo standard, ma alcune garantiscono maggiori chance di interoperabilità rispetto ad altre. Queste modalità sono state promosse a "specifica" nello standard WS-I Basic Profile (l'ultima versione è disponibile all'indirizzo *http://www.ws-i.org/Profiles/BasicProfile-1.1.html)*.

4.1 PROBLEMI DI COMPATIBILITÀ?

Chi utilizza diverse tecnologie e tool di sviluppo si può imbattere in alcuni problemi di compatibilità tra Web Services. Questo può sembrare

PuntoInformatico | Libri

una contraddizione rispetto all'interoperabilità conclamata delle tecnologie di base. In effetti il problema non risiede nei formati di rappresentazione ma nel loro uso e nelle assunzioni che ciascun tool fa riguardo ai dati e ai formati utilizzati. In sostanza si tratta di incompatibilità tra tool e non di incompatibilità tra tecnologie. Infatti uno dei consigli più comuni è di creare prima il WSDL e poi il servizio e non viceversa (rinunciando pertanto a tutta una serie di tool automatici che permettono di scrivere il codice e poi generare il WSDL in automatico). Questa strada può portare a dei problemi: che fare quando l'interfaccia (ovvero il codice sottostante) evolve? Sfortunatamente questo accade quasi per la totalità dei progetti reali: l'interfaccia pensata inizialmente è, per forza di cose, un'interfaccia di massima, i cui dettagli sono soggetti al cambiamento. Al momento non c'è soluzione: ci si augura che questi problemi scompaiano (o perlomeno avvengano molto di rado) al maturare dei tool di sviluppo disponibili….

4.2 WS-I BASIC PROFILE IN DETTAGLIO

WS-I Basic profile è un tentativo di dettare delle regole di utilizzo delle tecnologie di base dei Web Services, al fine di minimizzare il rischio di scrivere applicazioni che manchino di interoperabilità. Questo non significa che garantisca, in un qualsivoglia modo, l'interoperabilità: semplicemente detta delle regole di "buon senso" al fine di minimizzare i possibili problemi. Ogni "regola" è testabile: questo significa che non sono mai regole astratte, ma verificabili anche da un programma automatico; difatti le specifiche sono accompagnata da un insieme di tool che verificano la conformità dei manufatti rispetto alle indicazioni del profilo. Tutte le sue specifiche sono a livello di application-layer; questo significa che evita di scendere negli strati inferiori dello stack OSI.

4.3 REQUISITI DI CONFORMITÀ

Il profilo si compone di requisiti di conformità (conformance requirements) che specificano regole che debbono essere seguite al fine di considerare il manufatto conforme alla specifica; ciascun requisito ha un codice nmerico, preceduto dalla lettera R; il codice viene seguito dalla descrizione del requisito:

```
R2303 A DESCRIPTION MUST NOT use Solicit-Response and
    Notification type operations in a wsdl:portType definition
```

Indica il requirement numero 2303. Ci sono anche i conformance target, ovvero i destinatari di conformità. Questi indicano i manufatti a cui i requisiti si riferiscono (manufatti, in questo contesto, possono essere messaggi SOAP, documenti WSDL, e così via).

4.4 PUNTI DI ESTENSIONE

Esistono poi i punti di estensione (extensibility points) che indicano punti in cui il profilo non dà indicazioni e possono portare a problemi di incompatibilità se non c'è un accordo tra gli attori che utilizzano i Web Services.Anche essi sono indicati con un codice numerico ma, questa volta, preceduto da una lettera E (come nel caso precedente, dopo il codice segue la descrizione del punto di estensione); per esempio:

```
E0017 - Schema annotations - XML Schema allows for annotations,
which may be used to convey additional information about data structures.
```

4.5 CONFORMANCE TARGET

Ecco i target di conformità della specifica:

```
< Messaging
    < SOAP Envelops
```

PuntoInformatico | Libri

< SOAP Processing Model (modo di gestire i faults, header obbligatori, …)
< SOAP Faults
< Uso di SOAP in http (codici di errore e di successo, binding, cookie, …)
< Service description
< Required description
< Struttura del documento
< types
< messages
< portTypes
< bindings
< SOAP Bindings
< Uso degli XML Schema
< Service pubblication e discovery
< Binding templates
< tModels
< Security
< Uso di HTTPS

4.6 ALCUNE CARATTERISTICHE

Le sue specifiche, di particolare interesse, sono:

a) non utilizzare i SOAP Encoding;
b) è richiesto almeno il binding Http per SOAP;
c) è richiesto che eventuali messaggi SOAP di tipo Fault usino lo status 500 in risposta alle richieste Http;
d) ogni richiesta Http deve avvenire utilizzando il metodo POST;
e) L'interfaccia del Web Service deve essere descritta utilizzando il WSDL versione 1.1;
f) il tipo di SOAP Binding deve essere rpc/literal oppure document/literal;
g) non utilizzare gli stili Solicit/Response e Notification per le operazioni

h) è richiesto il supporto di UDDI versione 2.0

Un interessante documento di sintesi è reperibile alla pagina *http://www.ibm.com/developerworks/webservices/library/ws-basicprof.html*.

È importante sapere che esistono tool che verificano la conformità dei documenti WSDL alle specifiche del WS-I Basic Profile. Tali tool, insieme a documenti e altri "deliverables", sono resi disponibili dallo stesso sito WS-I Organization alla pagina *http://www.ws-i.org/deliverables/index.aspx*.

4.7 WS-I TESTING TOOLS

Come già accennato il WS-I Basic Profile è un insieme non solo di direttive ma anche di tool; si può eseguire il download dei WS-I Testing Tools alla pagina *http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools*.

4.8 DOWNLOAD E INSTALLAZIONE

Si esegua il download del file SSBP_Java_Tools-BdAD.zip, contenente i tool per Java. Basta eseguire l'estrazione dell'archivio e si presenta una struttura come quella mostrata in **Figura 4.2**

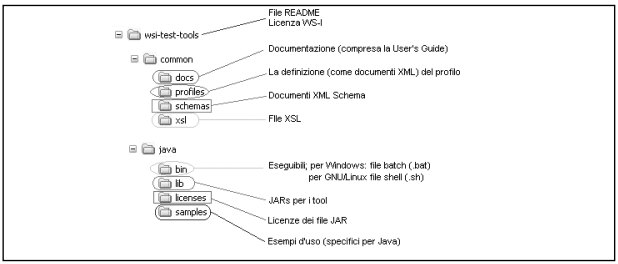

Figura 4.2: Struttura dei tool sul file system.

Nella cartella common/profile trovano posto i documenti TAD: Test As-

sertion Document, che guidano i tool nella verifica delle asserzioni del profilo. I tool installati sono due: il Monitor e l'Analyzer. Il primo serve ad intercettare i messaggi tra un utilizzatore ed un servizio Web al fine di memorizzare i messaggi inviati ed eseguirne il log. Il secondo verifica la conformità degli artefatti rispetto al profilo. Per esempio è in grado di analizzare i messaggi provenienti dal monitor, ma anche verificare e analizzare documenti WSDL che definiscono un servizio.

4.9 FILE DI CONFIGURAZIONE DI ANALYZER

Dalla cartella **java/samples** si possono utilizzare (personalizzandoli) alcuni file di esempio. In particolare si può personalizzare il seguente file: **analyzerConfig.xml**; esso contiene tutte le direttive che specificano al tool Analyzer dover reperire le diverse informazioni (file da analizzare, file di log da usare, tipo di report da generare e così via); esistono anche due file simili (**analyzerConfigServiceLocation.xml** e **analyzerConfigUDDI.xml**, che fanno la stessa cosa ma che reperiscono i file da analizzare in maniera diversa: uno lo fa da una locazione remota, l'altro attraverso un registro UDDI);

4.10 ESECUZIONE DI ANALYZER

Una volta configurato a dovere il proprio file di configurazione si può eseguire il file batch java\bin\Analyzer.bat. Per farlo ci si posizioni, con una console dei comandi, nella cartella **java** e si digiti (per Windows):

```
.\bin\setenv.bat
.\bin\Analyzer.bat -config samples\analyzerConfig.xml
```

Il primo file batch imposta le opportune variabili d'ambiente, il secondo esegue il tool usando il file di configurazione specificato. Nella cartella **java** viene creato il file **report.xml**: faccendovi doppio click vie-

ne aperto e visualizzato il report che riporta se il test di conformità è andato a buon fine oppure no (indica anche i dettagli relativi a ciascun requirement!); in (**Figura 4.3)** si vede che, come c'era da aspettarsi, il file di esempio è conforme al profilo (attenzione: affinché funzioni correttamente l'analisi dell'esempio è necessario avere una connessione ad Internet attiva).

Figura 4.3: il report di Analyzer per il file di esempio

4.11 CONFIGURAZIONE E ANALISI SU PRIMOWS

Non resta che creare un opportuno file di configurazione per analizzare il primo esempio di Web Service realizzato nel Capitolo 2 (primoWS). Si crei **java\libro\analyzerConfigPrimoWS.xml** e vi si inserisca il seguente documento xml (evidenziate le differenze rispetto al file di partenza):

```
<?xml version="1.0" encoding="UTF-8"?>
<wsi-analyzerConfig:configuration name="Analyzer Configuration"
   xmlns:wsi-analyzerConfig=
    "http://www.ws-i.org/testing/2004/07/analyzerConfig/">
 <wsi-analyzerConfig:description>
   Configura il Basic Profile Analyzer.
```

PuntoInformatico Libri

```
 </wsi-analyzerConfig:description>
 <wsi-analyzerConfig:verbose>false</wsi-analyzerConfig:verbose>
 <wsi-analyzerConfig:assertionResults
  type="all" messageEntry="true" failureMessage="true"/>
 <wsi-analyzerConfig:reportFile replace=
"true" location="libro/reportPrimoWS.xml">
   <wsi-analyzerConfig:addStyleSheet
                                                    href="../common/xsl/report.xsl"
     type="text/xsl"/>
 </wsi-analyzerConfig:reportFile>
 <wsi-analyzerConfig:testAssertionsFile>
   ../common/profiles/SSBP10_BP11_TAD.xml
 </wsi-analyzerConfig:testAssertionsFile>
 <wsi-analyzerConfig:logFile correlationType="endpoint">
   libro/log.xml
 </wsi-analyzerConfig:logFile>
 <wsi-analyzerConfig:wsdlReference>
   <wsi-analyzerConfig:wsdlElement type="port"
       parentElementName="primoWSService"
       namespace="http://127.0.0.1:8080/axis/primoWS.jws">
     primoWS
   </wsi-analyzerConfig:wsdlElement>
   <wsi-analyzerConfig:wsdlURI>
     http://127.0.0.1:8080/axis/primoWS.jws?wsdl
   </wsi-analyzerConfig:wsdlURI>
 </wsi-analyzerConfig:wsdlReference>
</wsi-analyzerConfig:configuration>
```

Si copi il file **java\samples\log.xml** in **java\libro\log.xml** e si eseguano i comandi:

```
.\bin\setenv.bat
.\bin\Analyzer.bat -config libro\analyzerConfigPrimoWS.xml
```

L'output sulla console è un resoconto dei file di configurazione coinvolti (**Figura 4.4**). Il "vero" risultato dell'esecuzione dell'ultimo comando è il file **java\libro\reportPrimoWS.xml**: si faccia doppio click per visualizzarne il contenuto; il test è fallito (`failed`)! Analizzando il report si scopre anche quale asserzione non è verificata: è la **BP2406** (**Figura 4.5**).

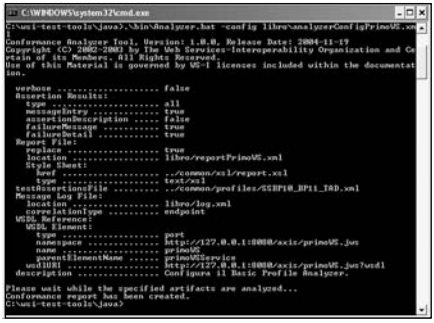

Figura 4.4: risultato sulla console.

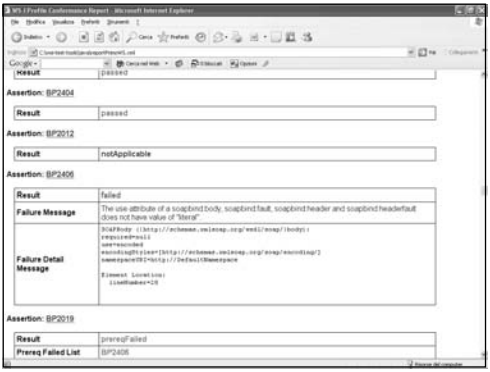

Figura 4.5: il report e l'asserzione non verificata..

Che significa **BP**? Esso sta per **Basic Profile**, ed è la specifica che detta l'asserzione (ci sono asserzioni che iniziano con la sigla **SSBP**: essa

PuntoInformatico Libri

sta per **Simple SOAP Basic Profile**, si veda **[WS-I-SSBP]**. Una caratteristica molto utile del report è che cliccando sul codice dell'asserzione, si apre il **file SSBP10_BP11_TAD.xml** e viene visualizzata una tabella che riassume il tipo di test e le asserzioni a cui il test afferisce, nonché il testo descrittivo del contesto e del tipo di requisiti, al fine di poter verificare il suo contenuto (**Figura 4.6)**.

Figura 4.6: il testo dell'asserzione non verificata.

In questo caso si può verificare che il **WSDL** del servizio analizzato non ha l'attributo **"literal"**: difatti l'uso all'interno dei messaggi è **"encoded"** e questo non è ammesso dal profilo (che invece ammette solo l'uso literal; pertanto gli stili ammessi sono, come già detto, solo **document/literal ed rpc/literal**).

BIBLIOGRAFIA

[AAVV, 2003] *"Building Interoperable Web Services: WS-I Basic Profile 1.0"* , Microsoft Press, 2003, disponibile in PDF dalla pagina http://msdn.microsoft.com/webservices/building/interop/default.aspx?pull=/library/en-us/dnsvcinter/html/wsi-bp_msdn_landingpage.asp

[WS-I-BP] *http://www.ws-i.org/Profiles/BasicProfile-1.1.html*

[WS-I-SSBP] *http://ws-i.org/Profiles/SimpleSoapBindingProfile-1.0-2004-08-24.html*

PuntoInformatico | Libri

REALIZZARE UN WSDL PARTENDO DA ZERO!

Oramai si dovrebbero avere le idee piuttosto chiare sulle tecnologie coinvolte e si dovrebbe avere dimestichezza con l'uso degli strumenti di base. È questo il momento di introdurre un esempio un po' più complesso, che possa avere una certa rilevanza pratica, pur senza appesantirlo di dettagli inutili. Ci si porrà il problema di rendere l'esempio compatibile con la specifica **WS-I Basic Profile 1.1.**

5.1 L'APPLICAZIONE DI ESEMPIO

Si supponga di voler creare un Web Service che consente ad un'azienda di vendere propri prodotti. Questi prodotti saranno composti da alcuni attributi, quali codice, descrizione, prezzo (e relativa valuta). I client che necessitano di effettuare degli ordini potrebbero agire in due fasi: nella prima verificano la disponibilità della quantità desiderata dei prodotti di cui hanno bisogno, ricevendo le quantità effettivamente disponibili. Verificato che le quantità disponibili siano comunque utili, confermano l'ordine. Tra la verifica della disponibilità e la conferma dell'ordine è necessario poter mantenere "prenotati" i prodotti di cui il server ha dichiarato la disponibilità.Ovviamente, di tanto in tanto (ma nulla esclude che possa essere fatto ad ogni richiesta) il client avrà la necessità di reperire la lista aggiornata dei prodotti disponibili. Siccome anche le prestazioni sono importanti, è troppo oneroso trasferire tutti i dati di tutti gli articoli tra un'invocazione e la successiva: è molto più conveniente spedire solo le modifiche (talvolta queste si chiamano il "delta" rispetto alla situazione precedente). Ovviamente è necessario tenere traccia della data dell'ultimo aggiornamento al fine di reperire solo quei dati modificati da allora (naturalmente è un problema del fornitore del servizio quello di rendere coerente l'erogazione del "delta" opportuno!). Questi i requisiti applicativi. Si vedrà come realizzare opportuni servizi che li soddisfano.

5.2 PRIMA L'UOVO O LA GALLINA?

La domanda da porsi è, ancor prima di iniziare a realizzare un Web Service, cosa si crea per primo? Le alternative sono due:

1. si crea prima il codice e poi si genera il WSDL che lo descrive attraverso un tool automatico;
2. si crea prima il WSDL che descrive il servizio e poi si generano gli stub e gli skeleton dei servizi che lo implementano (sempre grazie ad un opportuno tool).

La scelta non è poi così banale. Quasi tutti hanno una certa conoscenza del linguaggio di programmazione (sia esso Java, .NET o altro) e poca o nulla per il WSDL. In questi casi la scelta diviene quasi obbligata: si creano le classi e poi il WSDL.
Questo approccio, molto pragmatico, ha lo svantaggio di legarsi troppo all'implementazione dei servizi Web e dei tool che generano il WSDL. Il rischio (concreto) è quello che il WSDL generato non sia sufficientemente generale e si abbiano problemi di interoperabilità. È come se tutti i tool parlassero un dialetto dell'italiano: tutti riescono a capire l'italiano comune, ma non è detto che fra loro si possano parlare. Far generare il WSDL da un tool rischia di generare un documento in un "dialetto", incomprensibile a qualcun altro. Scriverlo a mano, seguendo le regole, equivale a scriverlo in italiano ed essere sicuri che ogni tool potrà leggerlo e interpretarlo correttamente.
Per questo la strada migliore è scrivere il WSDL per primo, anche se questo comporta uno studio approfondito del linguaggio e dei modi di scrivere documenti WSDL.
Questo processo è sicuramente lungo (più lungo di una generazione automatica, per lo meno), soggetto ad errori (scrivere in XML, con regole sintattiche e semantiche del WSDL, non è proprio la cosa più facile) e, in definitiva, richiede un notevole sforzo. Nel nostro caso questo sforzo è necessario e lo intraprendiamo da subito.

Il WSDL è, per definizione, un "contratto" che il server espone affinché un client possa far uso dei suoi servizi. In quest'ottica è fondamentale capire che tale contratto deve essere il più possibile chiaro "per tutti" (in questo caso gli interlocutori sono tutti i possibili client, sviluppati nelle diverse tecnologie e con diversi framework). Se si è nella condizione di dire che, in fondo, tale contratto non è poi così vincolante in quanto si ha completo controllo tanto del server quanto del client (e quindi si è in grado di imporre l'uso di uno strumento in ambo i casi) probabilmente la scelta di utilizzare i Web Services è poco ottimale: esistono altre scelte "proprietarie" più performanti e più stabili.

5.3 LE REGOLE DA SEGUIRE

Prima di realizzare un WSDL partendo da un documento vuoto è necessario avere le idee ben chiare su cosa si vuole ottenere. Per prima cosa è necessario aver ben chiare le tecnologie di base (XML Schema, namespace, SOAP e il WSDL stesso, ovviamente).
Inoltre è importante studiare e capire eventuali standard da seguire. Nel nostro caso, in cui è importante l'interoperabilità, è necessario conoscere almeno la specifica WS-I Basic Profile.
Inoltre è bene utilizzare lo stile document/literal. Rispetto allo stile rpc/literal è maggiormente supportato dai diversi tool (primo fra tutto Axis).

5.4 WSDL E XML SCHEMA CON SOA EDITOR

Realizzare un WSDL senza l'aiuto di uno strumento grafico è un compito piuttosto difficile e pieno di insidie. In commercio esistono numerosi tool, alcuni dei quali gratuiti. È il caso di SOA Editor, della CapeClear. Esso è scaricabile dalla *http://www.capescience.com/soa/download.shtml*. Nonostante sia gratuito (a paga-

mento c'è una versione più sofisticata dello strumento) ha tutto l'occorrente sia per realizzare un WSDL che per associarvi uno XML Schema. Una nota sui nomi (di tutti gli elementi del WSDL): si è scelto di utilizzare nomi in lingua inglese in quanto è verosimile che i client che faranno uso del Web Service siano internazionali. Se questo non fosse un requisito si può scegliere di utilizzare nomi italiani, più significativi in un contesto nazionale e non internazionale. Inizialmente è necessario dare un nome al WSDL e inserire il target namescape. Come si nota dalla (**Figura 5.2)**, il nome assegnato per l'esempio è ProductsExampleWS, mentre il namespace è http://ivenuti.altervista.org/ProductsExampleWS.wsdl (l'url va inserita a mano, il nome del file WSDL viene completato man mano che si digita il nome); si lascino selezionate tutte le check box sottostanti le due caselle di testo.

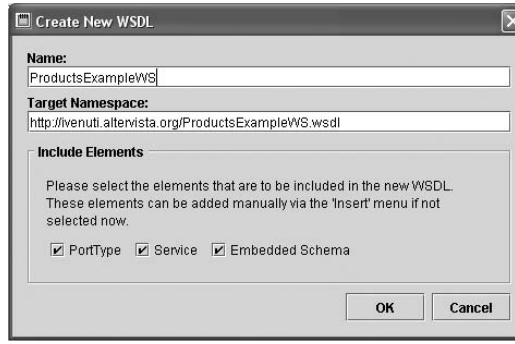

Figura 5.2: Scelte iniziali per il WSDL da creare

5.5 DEFINIRE LE OPERAZIONI

Si eliminino l'operazione esistente (di default) e i due messaggi; in questo modo si parte da una situazione "pulita". Le tre operazioni da inserire sono:

1) productsList

2) productsOrder
3) productsOrderConfirmation

Si scelga **Insert > Operation**. Ora si può procedere a definire la prima operazione. Posizionandosi sulla casella di testo **Name**, si digiti il nome del primo metodo. Sulla parte destra dell'editor, oltre a Name, trovano posto la casella di testo **Documentation** (dove, opzionalmente, va inserito un commento che serve a documentare l'uso dell'operazione; benché opzionale è sempre bene riempire anche questo campo) e una casella di riepilogo in cui si definisce il tipo (**Type**) di operazione (i valori possibili sono **Request/Response**, **One way**, **Notification**, Solicit/Response). Le tre operazioni dell'esempio saranno tutte di tipo Request/Response.

NOTA Si ricordi che WS-I Basic Profile non permette operazioni di tipo Notification né di tipo Solicit/Response (pertanto si sconsiglia in ogni caso il loro uso).

Sotto questi controlli ci sono tre schede chiamate, rispettivamente, `Messages, Parameters` e `Faults`.
Nella prima scheda si premano, per ogni operazione, i due pulsanti `New Message` (uno associato alla Request e una alla Response): questo porta a generare due nuovi messaggi chiamati tns:nomeOperazioneRequest e tns:nomeOperazioneResponse (**Figura 5.4** un esempio).Sempre agendo sul menu Insert > Operation si inseriscano altre due operazioni. Inserite le operazioni (definendo, come fatto per la prima, nome, descrizione e tipo e creando due messaggi per ognuna). Alla fine si ha una situazione come quella mostrata in (**Figura 5.3)** per quanto concerne le operazioni e si hanno i messaggi mostrati in (**Figura 5.4)**. Ci si sposti sulla pagina *Parameters*. Qui, agendo sul pulsante *Add*, è possibile inserire tutti i parametri delle operazioni. Prima però è necessario definire i tipi di dato necessari ad essere inseriti come parametri: infatti i tipi predefiniti non sono adatti a descrivere entità complesse

come i prodotti e gli ordini che si vogliono gestire nell'esempio (si ritornerà su questa pagina in seguito).
Infine la pagina *Faults* contiene eventuali errori sollevati dalle operazioni. Per l'esempio proposto non si immetteranno valori nei campi di inserimento di questa pagina.

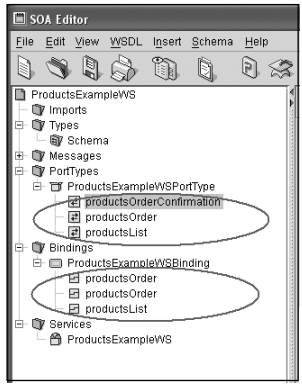

Figura 5.3: le tre operazioni

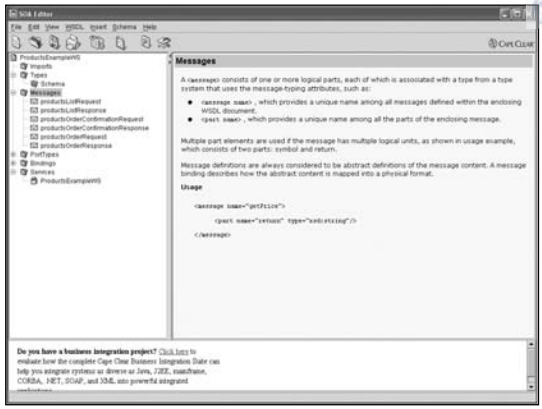

Figura 5.4: due nuovi messaggi per ogni operazione

PuntoInformatico Libri

5.6 DEFINIRE I TIPI

Posizionandosi, sul menu a sinistra, su *Types*, è possibile definire i tipi di dati del documento WSDL. In particolare, è possibile associare uno XML Schema che specifica tipi di elementi ed eventuali vincoli sui valori che tali elementi possono assumere. Per definire nuovi tipi si agisce sul menu *Schema* presenta nella barra dei comandi posta in alto dell'editor (**Figura 5.5**). Inizialmente lo schema contiene solo l'intestazione:

Figura 5.5: il menu che permette di personalizzare l'XML Schema del WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema
    targetNamespace="http://ivenuti.altervista.org/
            ProductsExampleWS.xsd1"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/enco
            ding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd= "http://ivenuti.alteravista.org/
            ProductsExamplesWS.xsdl
</xsd:schema>
```

Questo schema verrà riempito con i tipi definiti nel seguito. Si inizi con il definire il tipo Product: esso rappresenta un singolo prodotto. Per farlo si scelga Schema > Insert Complex Type.
Si può notare, dalla (**Figura 5.6**), come si è definito il tipo come tipo complesso (ComplexType) in cui l'ordine degli elementi è quello specificato (content model di tipo sequence), e con i campi evidenziati in (**Figura 5.6**).

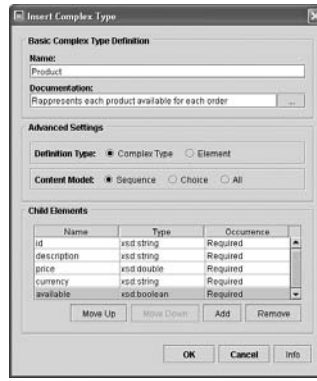

Figura 5.6: il tipo di dato Product.

PuntoInformatico Libri

Lo schema generato è il seguente:

```
<xsd:complexType name="Product">
  <xsd:annotation>
    <xsd:documentation>
      Rappresents each product available for each order
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" name="id"
            type="xsd:string"/>
      <xsd:element maxOccurs="1" minOccurs="1" name="description"
```

```
            type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="price"
            type="xsd:double"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="currency"
            type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="available"
            type="xsd:boolean"/>
    </xsd:sequence>
</xsd:complexType>
```

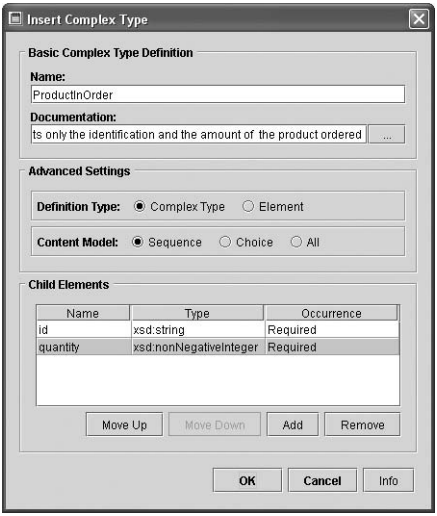

Figura 5.7: il tipo di dato ProductInOrder.

Poi si può definire il tipo ProductInOrder, che conterrà unicamente l'identificativo del prodotto presente nell'ordine e la quantità ordinata (**Figura 5.7**). In questo caso lo schema generato è:

```
<xsd:complexType name="ProductInOrder">
    <xsd:annotation>
```

```
      <xsd:documentation>
        Rappresents only the identification and the amount of
        the product ordered
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" name="id"
          type="xsd:string"/>
      <xsd:element
        maxOccurs="1"
        minOccurs="1"
        name="quantity"
        type="xsd:nonNegativeInteger"/>
    </xsd:sequence>
  </xsd:complexType>
```

Infine non resta che definire due tipi di dato che sono array dei tipi di dato appena definiti.Una strada possibile è quella di selezionare *Schema > Insert SOAP Encoded Array*. Eppure questa scelta è sconsigliata in quanto, come si è visto introducendo il WS-I Basic Profile, tale specifica vieta l'utilizzo di tipi SOAP Encoded. Per questo motivo è necessario definire un nuovo tipo in cui la voce *Occurence* è di tipo *Multiple*. In (**Figura 5.8)** la definizione del tipo ArrayOfProduct. Ecco l'XML Schema che definisce ArrayOfProduct:

Figura 5.8: il tipo di dato ArrayOfProduct.

```
<xsd:complexType name="ArrayOfProduct">    <xsd:annotation>
      <xsd:documentation>An array of products</xsd:documentation>
   </xsd:annotation>    <xsd:sequence>        <xsd:element
      maxOccurs="unbounded"            minOccurs="1"            name="item"
      type="xsd1:Product"/>    </xsd:sequence></xsd:complexType>
```

In maniera simile si definisce il tipo ArrayOfProductInOrder (**Figura 5.9**) e l'XML Schema corrispondente è:

Figura 5.9: il tipo di dato ArrayOfProductInOrder.

```
<xsd:complexType name="ArrayOfProductInOrder">
   <xsd:annotation>
      <xsd:documentation>An array of Products in order</xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element
         maxOccurs="unbounded"
         minOccurs="1"
         name="item"
         type="xsd1:ProductInOrder"/>
```

```
  </xsd:sequence>
</xsd:complexType>
```

Sempre secondo le specifiche WS-I Basic Profile è necessario inserire anche tanti elementi quanti sono i messaggi utilizzati (**Figura 5.10**). I messaggi corrispondono alle request e ai response di ogni operazione. L'operazione productsList avrà come response un elemento che racchiude la data dell'ultimo aggiornamento (**Figura 5.10**):

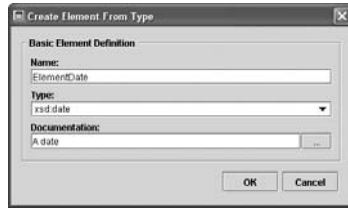

Figura 5.10: Un elemento di tipo ElementDate.

```
<xsd:element name="ElementDate" type="xsd:date">
  <xsd:annotation>
    <xsd:documentation>A date</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

La response dello stesso metodo conterrà un array con tutti i prodotti disponibili (**Figura 5.11**):

```
<xsd:element name="ElementArrayOfProduct"
    type="xsd1:ArrayOfProduct"> <xsd:annotation> <xsd:documentation>
      An element of type ArrayOfProduct
    </xsd:documentation>   </xsd:annotation></xsd:element>
```

Quando si vuole prenotare un ordine si avrà la necessità di un elemento con ArrayOfProductInOrder:

PuntoInformatico Libri

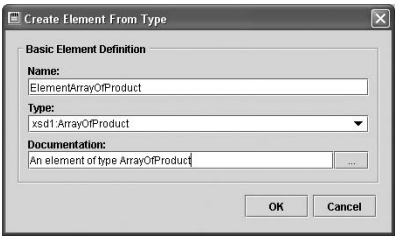

Figura 5.11: Un elemento di tipo ElementArrayOfProduct.

```
<xsd:element
   name="ElementArrayOfProductInOrder"
   type="xsd1:ArrayOfProductInOrder">
   <xsd:annotation>
     <xsd:documentation>
       An element of products used in orders
     </xsd:documentation>
   </xsd:annotation>
</xsd:element>
```

Il risultato di un ordine è sia un array di prodotti che un token per l'eventuale successiva conferma dell'ordine; per questo motivo vanno definiti un nuovo tipo complesso (**Figura 5.13**) e poi un elemento che lo racchiude (**Figura 5.14**):

```
<xsd:complexType name="OrderWithConfirmation">
   <xsd:sequence>
      <xsd:element
         maxOccurs="1"
         minOccurs="1"
         name="order"
         type="xsd1:ArrayOfProductInOrder"/>
      <xsd:element maxOccurs="1" minOccurs="1" name="magic"
type="xsd:string"/>
```

```
  </xsd:sequence>
</xsd:complexType>
<xsd:element
  name="ElementOrderWithConfirmation"
  type="xsd1:OrderWithConfirmation">
  <xsd:annotation>
    <xsd:documentation>
      An element for an order to be confirmed
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Si creano anche due elementi aggiuntivi: uno per contenere una stringa (magic) e uno per contenere un valore booleano:

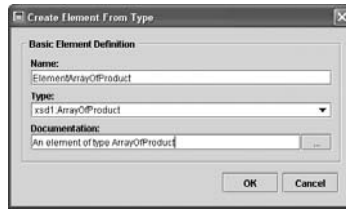

Figura 5.13: Un nuovo tipo complesso: OrderWithConfirmation.

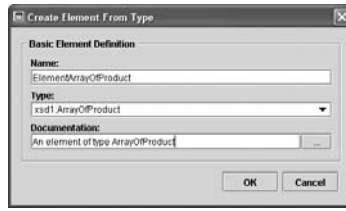

Figura 5.14: Un elemento di tipo ElementOrderWithConfirmation.

```
<xsd:element name="ElementMagic" type="xsd:string">
    <xsd:annotation>
```

```
      <xsd:documentation>
        String needed for the confirmation of an order
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name="ElementDone" type="xsd:boolean">
    <xsd:annotation>
      <xsd:documentation>
        Indicates id the operation is really done
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
```

5.7 DEFINIRE I MESSAGGI

Ora è possibile definire i messaggi che compongono sia la Request che la Response delle singole operazioni. Per farlo è possibile selezionare uno dei messaggi dal menu Messages (nella barra di navigazione di sinistra). In alternativa è possibile inserire i diversi parametri direttamente sulle operazioni nella pagina *Parameters*, che si è lasciata in sospeso all'inizio del capitolo.L'operazione productsList avrà due parametri: il primo (in ingresso) specifica la data dell'ultimo aggiornamento, il secondo (in uscita) restituisce la lista aggiornata dei prodotti ("delta" rispetto all'ultimo aggiornamento). I parametri inseriti (e i relativi tipi) sono mostrati in (**Figura 5.16)** e si riferiscono agli opportuni elementi definiti nello schema XML.ProductsOrder conterrà un array di prodotti da ordinare (in ingresso) e uno di quelli effettivamente disponibili (in uscita); in più ci sarà un token particolare che permetterà al client di confermare (successivamente) l'ordine prenotato (**Figura 5.17**).In (Figura **5.18)** si possono vedere i parametri dell'operazione di conferma dell'ordine: in ingresso il token del metodo precedente, in uscita un flag che indica se, effettivamente, la richiesta è andata a buon fine oppure no.

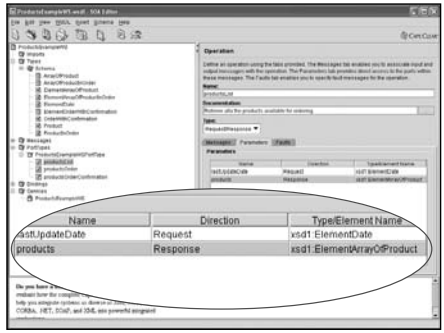

Figura 5.16: I parametri di productsList.

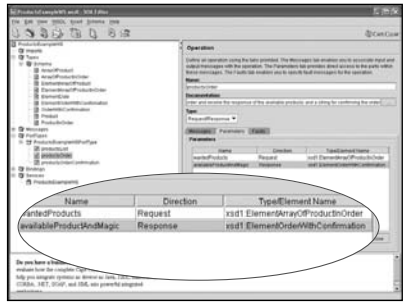

Figura 5.17: I parametri di productsOrder.

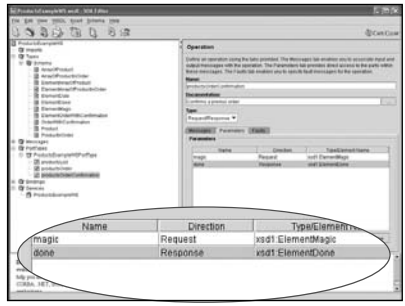

Figura 5.18: I parametri di productsOrderConfirmation.

5.8 BINDINGS

Selezionando, dal menu a sinistra, la voce *Bindings* si possono personalizzare i bindings del servizio. Innanzi tutto è necessario impostare la voce *Style* a *document*. Su transport si indica il trasporto utilizzato dal binding SOAP corrente (va bene lasciare il valore di default, http://schemas.xmlsoap.org/soap/http, per il binding al protocollo Http). Premendo il pulsante Add viene aggiunta la Parts opportuna (**Figura 5.19**).

5.9 SERVICES

Selezionando Services sul menu a sinistra è possibile impostare l'indirizzo dell'endpoint del servizio (address).
Ecco il WSDL completo (l'unico pezzo mancante, indicato con `<!--qui XML Schema-->`) è quello già presentato in precedenza per i tipi di dato definiti:

Figura 5.19: Uso del pulsante Add.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    name="ProductsExampleWS"
    targetNamespace="http://ivenuti.alteravista.org/
        ProductsExampleWS.wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://ivenuti.altervista.org/ProductsExampleWS.wsdl"
```

```
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsd1="http://ivenuti.altervista.org/
      ProductsExampleWS.xsd1">
<wsdl:documentation xmlns:wsdl=
      "http://schemas.xmlsoap.org/wsdl/">
</wsdl:documentation>
<wsdl:types>
      <xsd:schema
        targetNamespace="http://ivenuti.altervista.org/
                ProductsExampleWS.xsd1"
            xmlns:SOAP-ENC="http://schemas.xmlsoap.org/
                soap/encoding/"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsd1="http://ivenuti.altervista.org/
                ProductsExampleWS.xsd1">
            <xsd:import namespace="
                http://schemas.xmlsoap.org/wsdl/"/>
            <xsd:import namespace="
                http://schemas.xmlsoap.org/soap/encoding/"/>
<!— qui XML Schema —>
</xsd:schema>
</wsdl:types>
<wsdl:message name="productsOrderResponse">
<wsdl:part
      element="xsd1:ElementOrderWithConfirmation"
      name="availableProductAndMagic"/>
</wsdl:message>
<wsdl:message name="productsListResponse">
      <wsdl:part element="xsd1:ElementArrayOfProduct"
            name="products"/>
</wsdl:message>
```

PuntoInformatico Libri

```
<wsdl:message name="productsListRequest">
  <wsdl:part element="xsd1:ElementDate" name="lastUpdateDate"/>
 </wsdl:message>
<wsdl:message name="productsOrderConfirmationRequest">
  <wsdl:part element="xsd1:ElementMagic" name="magic"/>
 </wsdl:message>
<wsdl:message name="productsOrderConfirmationResponse">
  <wsdl:part element="xsd1:ElementDone" name="done"/>
 </wsdl:message>
<wsdl:portType name="ProductsExampleWSPortType">
  <wsdl:operation name="productsList">
    <wsdl:documentation xmlns:wsdl="
        http://schemas.xmlsoap.org/wsdl/">
        Retrieve alla the products available for ordering
        </wsdl:documentation>
<wsdl:input message="tns:productsListRequest"/>
    <wsdl:output message="tns:productsListResponse"/>
  </wsdl:operation>
  <wsdl:operation name="productsOrder">
    <wsdl:documentation xmlns:wsdl="
        http://schemas.xmlsoap.org/wsdl/">
        Makes a request of an order and receive the response
        of the available products and a string for
        confirming the order</wsdl:documentation>
<wsdl:input message="tns:productsOrderRequest"/>
    <wsdl:output message="tns:productsOrderResponse"/>
  </wsdl:operation>
  <wsdl:operation name="productsOrderConfirmation">
    <wsdl:documentation        xmlns:wsdl="
        http://schemas.xmlsoap.org/wsdl/">Confirms
        a previus order</wsdl:documentation>
<wsdl:input message="tns:productsOrderConfirmationRequest"/>
<wsdl:output message="tns:productsOrderConfirmationResponse"/>
```

```
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding
name="ProductsExampleWSBinding"
  type="tns:ProductsExampleWSPortType">
<soap:binding style="document" transport="
   http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="productsList">
    <soap:operation
soapAction="capeconnect:ProductsExampleWS:
   ProductsExampleWSPortType#productsList"/>
<wsdl:input>
       <soap:body parts="lastUpdateDate" use="literal"/>
      </wsdl:input>
      <wsdl:output>
       <soap:body parts="products" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="productsOrder">
<soap:operation soapAction="capeconnect:ProductsExampleWS:
   ProductsExampleWSPortType#productsOrder"/>
      <wsdl:input>
       <soap:body parts="wantedProducts" use="literal"/>
      </wsdl:input>
      <wsdl:output>
     <soap:body parts="availableProductAndMagic" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
<wsdl:operation name="productsOrderConfirmation">
<soap:operation
   soapAction="capeconnect:ProductsExampleWS:
        ProductsExampleWSPortType#productsOrderConfirmation"/>
<wsdl:input>
```

PuntoInformatico Libri

```
        <soap:body parts="magic" use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body parts="done" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="ProductsExampleWS">
<wsdl:port binding="tns:ProductsExampleWSBinding" name="
    ProductsExampleWSPort">
<soap:address location="
    http://localhost:8080/axis/ProductsExampleWS"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

5.10 TEST DI CONFORMITÀ

Una volta creato il WSDL è opportuno verificare che esso sia conforme alle specifiche WS-I Basic Profile.
Per far questo si esegue uno dei tool messi a disposizione: Analyzer (per la sua configurazione e modalità di esecuzione si riveda il Capitolo 4). Eseguito il test, il WSDL risulta conforme alle specifiche!

BIBLIOGRAFIA

[WS-I-BP] http://www.ws-i.org/Profiles/BasicProfile-1.1.html
[WSDL, 2001] "*Web Services Description Language (WSDL) 1.1*", W3C Note, http://www.w3.org/TR/wsdl

PuntoInformatico | Libri

AXIS IN DETTAGLIO

Axis è un progetto della Apache Software Foundation e prende origine dal progetto Apache SOAP: durante lo sviluppo di quest'ultimo è nata l'esigenza di una riscrittura dell'intero progetto per farlo evolvere ad un'architettura maggiormente modulare e con un meccanismo di parsing XML di tipo SAX. Visto l'impatto notevole delle nuove modifiche, sia in termini architetturali che di utilizzo, è stato abbandonato il nome di Apache SOAP ed è nata la prima versione di Axis. Nel momento in cui il libro viene scritto, l'ultima versione del progetto è la 1.2. I cambiamenti rispetto alla versione 1.1 sono notevoli, soprattutto per quanto riguarda l'interoperabilità con altri framework: tale aspetto è quello più importante visto che nel corso del libro illustreremo molte soluzioni sviluppate nei diversi linguaggi e utilizzando framework e ambienti di sviluppo eterogenei.

6.1 APACHE WEB SERVICES PROJECT

Axis è un SOAP engine: questo significa che è un framework che si concentra unicamente sulle problematiche della creazione di client e server per la gestione di messaggi SOAP. Eventuali estensioni (WS-Specifications, che verranno presentate nel Capitolo 10) non sono affrontate da Axis, ma sono facilmente integrabili utilizzando altre tecnologie, alcune delle quali sviluppate all'interno dell'Apache Web Services Project (http://ws.apache.org/), mentre Axis consiste in un insieme di tool per la generazione automatica di classi e in una libreria che "incapsula" in classi Java l'accesso alle tecnologie connesse ai Web Services.

6.2 UN ESEMPIO UN PO' PIÙ COMPLESSO

Si provi a scrivere la seguente classe:

```
package it.ioprogrammo.ws;
```

```
public class primoWS {
    public primoWS(){
        System.out.println("primoWS:costruttore...");
    }
    public String Metodo1(){
        System.out.println("primoWS:Metodo1...");
        return "Metodo1";
    }
    public String Metodo2(String chiSei){
        System.out.println("primoWS:Metodo2...; chiSei='"+chiSei+"'");
        return "Metodo2 risponde a "+chiSei;
    }
    public String Metodo3(){
        System.out.println("primoWS:Metodo3...");
        return "Metodo3";
    }
}
```

Essa è composta da 3 metodi: si supponga di voler eseguire il deploy e di esporre solo i primi due metodi. Già questo fa capire che non si può semplicemente compilare la classe e rinominarla con estensione JWS: tutti i tre metodi sarebbero esposti. Inoltre si dovrebbe rinunciare alla possibilità di creare la classe all'interno del package it.ioprogrammo.ws (ma, in fin dei conti, per un esempio così semplice questo non sarebbe un grosso problema). L'unica alternativa è creare un file WSDD ed eseguire un deploy.

6.3 LA CONFIGURAZIONE VIA WSDD

Le informazioni su ogni servizio sono ottenute da un file WSDD (Web Service Deployment Descriptor); esso è specifico di Axis e

consente di specificare, attraverso un file XML che segue opportune regole, come deve comportarsi il servizio Web una volta installato (la classe che si preoccupa di gestire i file WSDD è org.apache.axis.EngineConfiguration). L'elemento principale di un file WSDD è deployment e, per quanto concerne i servizi java, è standard:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
        xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
. . .
</deployment>
```

Al suo interno trova posto l'elemento service, i cui attributi sono il nome del servizio (name) e provider, mentre al suo interno è necessario specificare il nome della classe che implementa il servizio (className) e quali sono i metodi (della classe) abilitati ad essere esposti come operazioni (è possibile utilizzare il carattere * per indicare che tutti i metodi contenuti sono esposti all'esterno, mentre se si vuole abilitarne solo alcuni è necessario fornire una lista di nomi separata da punto e virgola o da spazi); per l'esempio proposto:

```
<service name="primoWSEsempio" provider="java:RPC">
  <parameter name="className"
value="it.ioprogrammo.ws.primoWS"/>
  <parameter name="allowedMethods" value="Metodo1 Metodo2"/>
 </service>
```

Dove risiede il file WSDD di Axis? Esso è presente nella cartella WEB-INF/ dell'applicazione (quindi, normalmente, sotto webapps/axis) e si chiama server-config.wsdd. Al suo interno è già presente l'elemento deployment e tutta una serie di servizi. Basterà aggiungere il servizio sopradescritto prima di tutti gli altri. Ovviamente la classe compilata dovrà esse-

re resa disponibile all'applicazione axis o tramite un file jar da posizionarsi sotto WEB-INF/lib/ oppure copiando la classe compilata in WEB-INF/classes/it/ioprogrammo/ws/. Per rendere attive le modifiche al file WSDD è necessario far ripartire l'istanza di Tomcat. Accedendo alla pagina *http://127.0.0.1:8080/axis/* e seguendo il link "List", il nuovo servizio Web apparirà tra quelli disponibili (e seguendo il link WSDL verrà mostrato il relativo file WSDL generato da Axis!). è importante osservare che solo Metodo1 e Metodo2 risultano operazioni invocabili sul nuovo servizio (**Figura 6.2**).

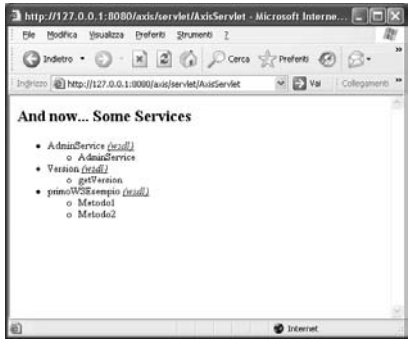

Figura 6.2: il nuovo servizio Web: primoWSEsempio.

6.4 SCRIVERE IL CLIENT

Per verificare il funzionamento del servizio Web appena pubblicato si può scrivere un client che accede alle sue operazioni. Si possono utilizzare le classi fornite da Axis sia per creare un'istanza di un servizio (classe Service) che per inizializzare una chiamata al servizio stesso (classe Call). Tale chiamata deve contenere almeno l'endpoint del servizio (specificato da una URL), il nome dell'operazione attraverso setOperationName (nell'esempio si testi Metodo1), eventuali parametri di ingresso (in questo caso non ce ne sono; se ci fossero si dovrebbe usare il metodo addParameter) e di ritorno (setReturnType); ecco la classe che esegue

il test di invocazione e stampa il risultato ottenuto:

```
package it.ioprogrammo.ws;
public class primoWSEsempioClient{
  public static void main(String [] args){
    try {
      org.apache.axis.client.Call call =
        (org.apache.axis.client.Call)
        ( new org.apache.axis.client.Service()
          ).createCall();
      call.setTargetEndpointAddress(
         new java.net.URL(
          "http://127.0.0.1:8080/axis/services/primoWSExample")
        );
      call.setOperationName(
         new javax.xml.namespace.QName(
           "http://ws.ioprogrammo.it",
           "Metodo1")
       );
      call.setReturnType(
        org.apache.axis.encoding.XMLType.XSD_STRING );
      System.out.println("Risultato: " +
       (String)call.invoke( new Object[0] ));
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

6.5 ULTERIORI OPZIONI: LO SCOPE

Eseguendo il client precedente è possibile verificare sia la stampa del client che quelle relative al server. Queste ultime sono

ancora più interessanti per quanto riguarda il costruttore: esso viene invocato ogni volta che si esegue la chiamata al metodo. Ciò significa che un oggetto di tipo primoWS viene creato (e distrutto) ad ogni invocazione.In alcuni casi si potrebbe preferire che un solo oggetto venga istanziato e che esso soddisfi tutte le richieste; in questo caso è necessario specificare, nel file WSDD, lo `scope` e impostarlo al valore Application:

```
<\ name="primoWSEsempio" provider="java:RPC">
 <parameter name="scope" value="Application"/>
 ...
</service>
```

In alternativa è possibile utilizzare i valori di `Session` (nel caso si gestiscano le sessioni, viene creato un nuovo oggetto per ogni sessione) e `Request` (che è il valore di default).

6.6 USARE ADMINCLIENT

Anziché intervenire direttamente sul file server-config.wsdd di Axis, è possibile scrivere un file WSDD per ogni servizio di cui si vuole effettuare il deploy (comprensivo dell'elemento deploymnet) e poi invocare la classe `org.apache.axis.client.AdminClient` per eseguire il deploy su Axis; in questo modo il file server-config.wsdd viene aggiornato da Axis stesso evitando possibili errori e blocchi dell'intera installazione (infatti eventuali errori sul file wsdd sono segnalati dall'applicazione stessa):

```
java org.apache.axis.client.AdminClient deploySpecifico.wsdd
```

grazie ad AdminClient è anche possibile eseguire l'undeploy di un servizio.

Tecnico Informatico Libri

Per farlo basterà costruire un file wsdd con l'elemento undeployment e specificando il servizio di cui si vuole l'undeploy; nel caso del servizio di esempio esso sarà un file, per esempio undeployEsempio.wsdd, contenente:

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
 <service name="primoWSEsempio"/>
</undeployment>
```

E l'undeploy avviene invocando AdminClient in questo modo:

```
java org.apache.axis.client.AdminClient undeployEsempio.wsdd
```

Per valutare le altre caratteristiche dei file WSDD è necessario conoscere l'architettura specifica di Axis. Pertanto ora la si presenterà nel dettaglio…

6.7 L'ARCHITETTURA

Una qualsiasi classe può essere esposta da Axis come un Web Service. In pratica Axis fornisce un'infrastruttura, chiamata Axis Engine, per "mascherare" i dettagli relativi alla traduzione da/verso messaggi SOAP a classi Java; tale engine si potrebbe utilizzare senza entrare nel merito dei suoi dettagli (lo si vedrebbe come una "scatola nera" **Figura 6.3**).
Ovviamente per qualunque sviluppatore di applicazioni complesse diventa cruciale conoscere più da vicino l'architettura interna di Axis in maniera da trarne vantaggio e comprenderne più a fondo eventuali limitazioni o potenzialità.
Axis si basa sul concetto di catene (*chains*): ogni catena è vista come un insieme di handlers in cui il risultato di un handler è l'ingresso di quello successivo (architettura detta anche di tipo *pipe-line*) come mostrato in (**Figura 6.4)**.

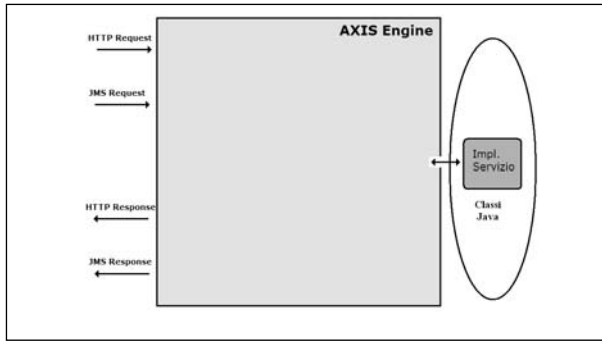

Figura 6.3: Axis visto come una "scatola nera".

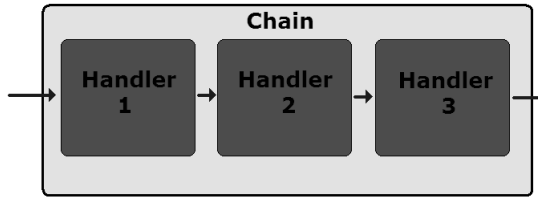

Figura 6.4: Una catena (chain) e i relativi handlers.

Axis può combinare un numero qualunque di catene in due flussi logicamente distinti: uno è la Request, ovvero il flusso che proviene da un generico client, l'altro è la Response, ovvero il flusso in uscita verso il client che ha inoltrato la chiamata. Questi flussi, inoltre, possono prevedere, a loro volta, catene specifiche per ciascun tipo di trasporto (infatti Axis, aderendo alle specifiche SOAP 1.2, non vincola l'utilizzo di un singolo tipo di trasporto, quale http, JMS, o altro). Inoltre ogni servizio può avere una propria catena (legata pertanto allo specifico servizio). In complesso l'architettura risultante è mostrata in (**Figura 6.5)**.

Quella descritta è l'architettura di un server realizzato con

Axis. Per quanto concerne i client (infatti Axis può essere utilizzato, come si è visto, anche per generare client di Web Services esistenti) tale architettura è speculare (**Figura 6.6**).

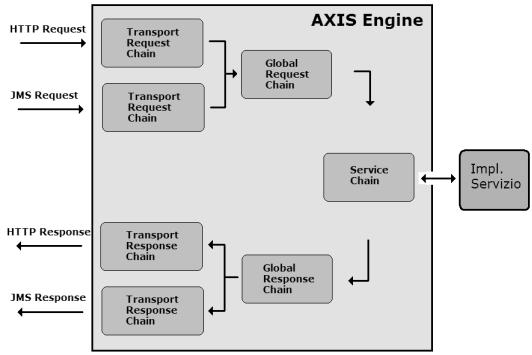

Figura 6.5: L'architettura di Axis.

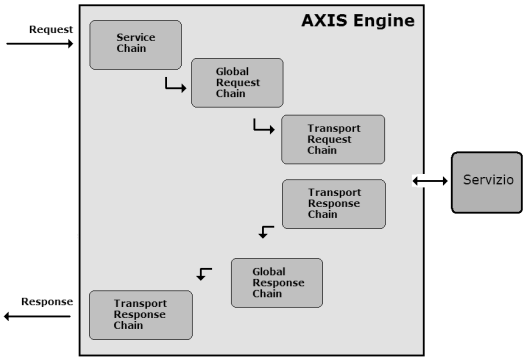

Figura 6.6: L'architettura di Axis (caso di un client).

6.8 UN ESEMPIO DI HANDLER

Un problema comune a tutte le applicazioni è quello di avere un indice di performance. Un modo molto semplice per realizzarlo è quel-

lo di "contare" il numero di millisecondi per ogni servizio. Si potrebbe inserire questo codice in ogni punto che si vuole monitorare ma, verosimilmente, è più che sufficiente monitorare l'esecuzione complessiva di ciascun metodo. Anziché inserire all'interno del codice le istruzioni di monitoring, si possono realizzare due handler: uno che viene invocato subito dopo la request e uno appena si inizia la response. Questo ha il vantaggio di rendere il codice di monitoring separato dal codice applicativo e giova alla flessibilità del suo deploy (può essere su un singolo servizio, su più o su tutti). Per realizzare un qualsiasi handler basta realizzare una classe che implementi l'interfaccia BasicHandler e il metodo invoke; ecco, per esempio, l'handler associato alla request:

```
package it.ioprogrammo.ws.handlers;
import org.apache.axis.*;
import org.apache.axis.handlers.*;
public class RequestLogger extends BasicHandler{
      public void invoke(MessageContext arg0) throws AxisFault {
   java.util.Date inizio = new java.util.Date();
   arg0.setProperty("startLogger", inizio);
  }
}
```

Esso non fa altro che memorizzare l'inizio dell'esecuzione in una proprietà del contesto associato al messaggio corrente (MessageContext). L'altro handler dovrà reperire tale proprietà ed eseguire la differenza sul tempo corrente:

```
public class ResponseLogger extends BasicHandler{
  public void invoke(MessageContext arg0) throws AxisFault {
   long lstartD = 0;
   long lendD = (new java.util.Date()).getTime();
   try {
```

```
    lstartD = ((java.util.Date) arg0.getProperty("startLogger")).getTime();
    }catch(Exception e) {
     e.printStackTrace();
    }
    System.out.println(arg0.getOperation().getName()+ " - "+
      (lendD-lstartD) + " msec"
  }
}
```

L'handler stampa il nome dell'operazione invocata e il tempo (in millisecondi) della sua esecuzione. Ovviamente anziché eseguire una semplice stampa sulla console, tale handler potrebbe contenere una logica più complessa (eseguire il log su un file o su database, ma anche verificare se la performance degrada nel tempo e così via).

6.9 COME INSERIRLO IN UNA CATENA

Per inserire un handler in una catena è necessario modificare il file `server-config.wsdd`. In particolare, per associare un handler a qualsiasi request o response si deve inserirlo nella sezione globalConfiguration; se si vuole inserirlo su uno specifico servizio lo si farà nella sezione service. Ecco come associare i due handler creati affinché rispondano ad ogni richiesta di servizio:

```
...
<globalConfiguration>
  ...
  <requestFlow>
   <handler type="java:it.ioprogrammo.ws.handlers.RequestLogger">
    <parameter name="scope" value="application"/>
```

```
  </handler>
 ...
 </requestFlow>
 <responseFlow>
  <handler type="java:it.ioprogrammo.ws.handlers.ResponseLogger">
   <parameter name="scope" value="application"/>
  </handler>
 ...
 </responseFlow>
</globalConfiguration>
...
```

6.10 USARE HANDLERS O FILTRI J2EE?

Si è detto un handler può essere associato anche ad un determinato tipo di trasporto. Si potrebbe, per esempio, pensare di associare un handler per verificare che un utente fornisca credenziali di tipo http Basic Authentication per autenticarlo ed autorizzarlo al servizio richiesto. Benché questo sia facilmente realizzabile, c'è una controindicazione: tale handler può essere utilizzato unicamente con Axis; non è pertanto facilmente portabile ad una qualunque applicazione J2EE.
In tale ambiente esiste un meccanismo analogo agli handlers: i filtri.
Ecco che nel caso di http basic authentication è preferibile utilizzare un filtro. La scelta di quando usare un filtro oppure un handler è soggettiva e va valutata caso per caso.
Di solito si preferisce realizzare un handler quando è necessario implementare handler simili per diversi tipi di trasporto: in questo caso l'architettura risulta più "pulita" se si realizzano vari handler il cui scopo è comune a tutti ma la cui realizzazione è dipendente dal tipo di trasporto a cui sono associati.

Libri Punto Informatico

ALTRE CARATTERISTICHE

6.11 INSERIRE UN PROPRIO WSDL

Axis genera "dinamicamente" un WSDL a partire dal servizio di cui si esegue il deploy. Nel caso si voglia inserire un proprio WSDL, ignorando quello generato da Axis, è necessario, subito dopo l'elemento service, inserire il seguente elemento:

```
<wsdlFile>nomeFile.wsdl</wsdlFile>
```

6.12 GLI "STILI" DISPONIBILI IN AXIS

Axis possiede quattro stili per i servizi deploytati: RPC, Document, Wrapped e Message.I primi due sono stati citati quando sono stati descritti i due possibili stili per scrivere messaggi SOAP; Wrapped in realtà è simile al document, ma anziché utilizzare un unico elemento per rappresentare l'intera struttura del body, utilizza un elemento per ogni sua parte. Message, infine, è lo stesso di Document ma non effettua il binding tra XML e strutture dati in Java (in pratica fornisce i documenti XML che poi il programmatore dovrà trattare),

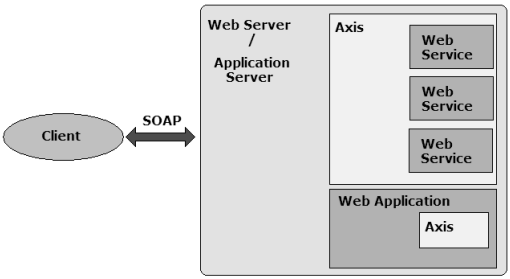

Figura 6.7: Axis come Web Server Container o embedded in una Web Application

mentre usando lo stile Document si effettua anche il binding.

6.13 COME UTILIZZARE AXIS

Axis può essere utilizzato in due modalità: o come Web Service Container oppure all'interno di una Web Application esistente. Nel primo caso esso fungerà, oltre che da framework per lo sviluppo, anche da ambiente dove eseguire deploy / undeploy di servizi Web. Nel secondo caso sarà di supporto per esporre, come servizi Web, funzionalità già esistenti (e sarà detto embedded all'interno dell'applicazione Web). L'esempio sarà sviluppato utilizzando Axis nella prima modalità.

6.14 TOOL "ESTERNI"

Java2WSDL, WSDL2Java sono due tool forniti con Axis per, rispettivamente, la creazione di file WSDL a partire da calssi Java e la creazione della struttura delle classi che implementano i servizi (come server o come client) specificati da un WSDL esistente.
Essi non sono utilizzati internamente da Axis ma rappresentano i tool che qualsiasi sviluppatore si troverà ad utilizzare (molto più di altre classi che realizzano dettagli implementativi "interni" ad Axis stesso).

6.15 REALIZZARE UN CLIENT!

Si utilizza il tool WSDL2Java anche per la generazione del client; tra le altre cose è possibile specificare il package in cui le classi saranno generate (di defaultsi seguirà il namescape indicato nel fie WSDL); per esempio ecco il comando per generare le classi client per il servizio descritto in ProductExampleWS.wsdl e all'interno del package it.ioprogrammo.ws:

```
java -classpath %AXISCLASSPATH%
 org.apache.axis.wsdl.WSDL2Java
 —package it.ioprogrammo.ws
 —verbose
 %WSDL_HOME%\ProductsExampleWS.wsdl
```

In (**Figura 6.8)** le classi generate. Non resta che scrivere un programma che faccia uso di tali classi; il programma presentato reperisce la lista di tutti i prodotti disponibili e li ordina tutti. Infine conferma l'ordine qualunque sia la disponibilità (la logica è semplificata per far vedere le iterazioni tra client e server):

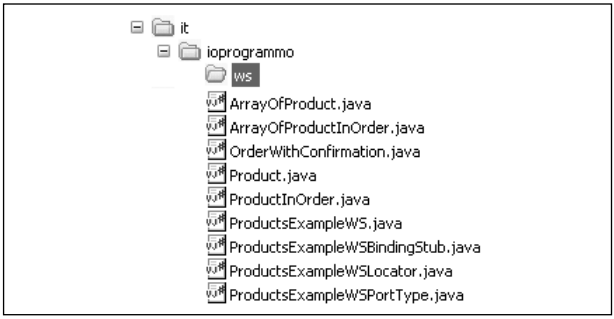

Figura 6.8: le classi per il client, come sono state generate da WSDL2Java.

```
package it.ioprogrammo.ws.client;
import java.net.URL;
import java.util.Date;
import org.apache.axis.types.*;
import it.ioprogrammo.ws.*;
public class testIt {
  public static void main(String[] args) {
    System.out.println("Running...");
    try {
     ProductsExampleWSLocator loc = new
```

```
                                        ProductsExampleWSLocator();
    ProductsExampleWSPortType pt =
      loc.getProductsExampleWSPort( new URL(
      "http://127.0.0.1:8080/axis/services/ProductsExampleWSPort"));
    ArrayOfProduct array = pt.productsList( new Date() );
    System.out.println("\nI prodotti disponibili:");
    stampaArray(array);
    ArrayOfProductInOrder ordine = ordinaTutti(array, 1);
    System.out.println("\nI prodotti ordinati:");
    stampaArray(ordine);
    OrderWithConfirmation risposta = pt.productsOrder(ordine);
    System.out.println("\nI prodotti prenotati:");
    stampaArray(risposta.getOrder());
   boolean done=pt.productsOrderConfirmation(risposta.getMagic());
    System.out.println("\nOrdine andato a buon fine? "+done);
    System.out.println("\nRiprova (stesso magic)...");
    done=pt.productsOrderConfirmation(risposta.getMagic());
    System.out.println("Ordine andato a buon fine? "+done);
  } catch (Exception e) {
    e.printStackTrace();
  }
  System.out.println("...finished!");
}
```

Ecco il metodo che effettua l'ordine di num elementi di ciascun prodotto passato come argmento:

```
private static ArrayOfProductInOrder ordinaTutti(
   ArrayOfProduct array, int num) {
 ArrayOfProductInOrder ris = new ArrayOfProductInOrder();
 if (array!=null) {
    Product[] pr = array.getItem();
    if (pr!=null) {
```

```
        ProductInOrder[] arrayProd = new ProductInOrder[ pr.length ];
        ris.setItem(arrayProd);
        for(int i=0; i<pr.length; i++) {
          arrayProd[i] = new ProductInOrder();
          arrayProd[i].setId( pr[i].getId() );
          arrayProd[i].setQuantity( new PositiveInteger(""+num));
        }
      }
    }
    return ris;
}
```

Infine ecco i metodi "ausiliari" per la sola stampa sulla console degli array di prodotti:

```
private static void stampaArray(ArrayOfProduct array) {
    if (array!=null) {
      Product[] pr = array.getItem();
      if (pr!=null) {
        for(int i=0; i<pr.length; i++)
          System.out.println(pr[i].getId()+", "+
            pr[i].getDescription()+", "+pr[i].isAvailable());
      }
    }
}
 private static void stampaArray(ArrayOfProductInOrder array) {
  if (array!=null) {
      ProductInOrder[] pr = array.getItem();
      if (pr!=null) {
        for(int i=0; i<pr.length; i++)
          System.out.println("( "+pr[i].getId()+", "+
            pr[i].getQuantity()+")");
      }
```

```
    }
  }
}
```

L'output di una possibile esecuzione del client presentato (utilizzando Eclipse). Sarebbe interessante conoscere i dettagli dei messaggi SOAP scambiati tra client e server: esiste uno strumento di Axis adatto a tale scopo: SOAPMonitor…

6.16 I MESSAGGI TRA CLIENT E SERVER

Si inizia a indagare sui dettagli dei messaggi "curiosando" cosa accade dietro le quinte tra client e server. Per farlo si abiliterà un programma di utilità, chiamato SOAPMonitor, distribuito insieme ad Axis ma che, per ragioni di sicurezza, non è attivo nell'installazione standard. Tale classe è presente in formato sorgente sotto la directory webapps/axis (e pertanto è presente anche sotto la webapps di Tomcat se si è copiata tale directory, come consigliato nel Capitolo 2). Per compilare la classe:

```
javac -classpath %AXIS_HOME%\lib\axis.jar SOAPMonitorApplet.java
```

e poi copiare i file risultanti, i .class visibili in (**Figura 6.10)**, sotto la directory webapps/axis/ di Tomcat.
Inoltre è necessario effettuare il deploy del servizio (compilando la classe si è abilitato la parte client, ora si deve abilitare la parte server del monitor). Ecco il file wsdd, che si può inserire in un file di nome deploy-monitor.wsdd:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
ava="http://xml.apache.org/axis/wsdd/providers/java">
 <handler name="soapmonitor"
```

```
    type="java:org.apache.axis.handlers.SOAPMonitorHandler">
  <parameter name="wsdlURL"
    value="/axis/SOAPMonitorService-impl.wsdl"/>
  <parameter name="namespace"
    value="http://tempuri.org/wsdl/2001/12
     /SOAPMonitorService-impl.wsdl"/>
  <parameter name="serviceName" value="SOAPMonitorService"/>
  <parameter name="portName" value="Demo"/>
 </handler>
 <service name="SOAPMonitorService" provider="java:RPC">
  <parameter name="allowedMethods" value="publishMessage"/>
  <parameter name="className"
    value="org.apache.axis.monitor.SOAPMonitorService"/>
  <parameter name="scope" value="Application"/>
 </service>
</deployment>
```

```
<DIR>          .
<DIR>          ..
         2.876 EchoHeaders.jws
         9.912 fingerprint.jsp
        15.524 happyaxis.jsp
         7.335 i18nLib.jsp
           603 index.html
         2.851 index.jsp
        58.258 SOAPMonitorApplet.java
         2.034 StockQuoteService.jws
<DIR>          WEB-INF
        11.900 SOAPMonitorApplet$SOAPMonitorPage.class
         1.470 SOAPMonitorApplet$SOAPMonitorData.class
         4.612 SOAPMonitorApplet$SOAPMonitorTableModel.clas

         4.741 SOAPMonitorApplet$ServiceFilterPanel.class
         5.028 SOAPMonitorApplet$SOAPMonitorFilter.class
         2.087 SOAPMonitorApplet$SOAPMonitorTextArea.class
         2.922 SOAPMonitorApplet.class
e       132.153 byte
ectory  11.881.267.200 byte disponibili

ps\axis>_
```

Figura 6.10: il risultato dell'esecuzione del client (console di Eclipse).

Ecco come fare il deploy del servizio:

```
java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient
    lhttp://localhost:8080/axis/services/AdminService deploy-monitor.wsdd
```

Infine è necessario indicare, sul file server-config.wsdd, quali servizi devono essere monitorati inserendo il seguente codice XML dopo l'elemento service:

```
<requestFlow>
  <handler type="soapmonitor"/>
</requestFlow>
<responseFlow>
  <handler type="soapmonitor"/>
</responseFlow>
```

Ovviamente se si vuole monitorare solo la request o solo la response, basterà inserire solo uno dei due elementi indicati.
Si supponga di inserire tali informazioni sia per il servizio primoWSEsempio che per il servizio ProductsExampleWSPort:

```
<service name="ProductsExampleWSPort"
 provider="java:RPC" style="document" use="literal">
 <requestFlow>
  <handler type="soapmonitor"/>
 </requestFlow>
 <responseFlow>
  <handler type="soapmonitor"/>
 </responseFlow>
…
</service>
 <service name="primoWSEsempio" provider="java:RPC">
 <requestFlow>
  <handler type="soapmonitor"/>
 </requestFlow>
 <responseFlow>
  <handler type="soapmonitor"/>
 </responseFlow>
```

PuntoInformatico Libri

Ora basterà invocare l'url *http://127.0.0.1:8080/axis/SOAPMonitor* per vedere in esecuzione il monitor. Tale monitor registra tutte le interazioni SOAP tra il server ed eventuali client.Ricapitolando, ecco i passi necessari per rendere attivo il monitor:

compilare il file SOAPMonitorApplet.java e mettere i file risultanti nella webapp axis;

eseguire il deploy del servizio (lato server) del monitor;

inserire gli opportuni tag ai servizi da monitorare.

BIBLIOGRAFIA

[Chappell & Jewell, 2002] "*Java Web Services*", D.A.Chappell, T. Jewell, HopsLisbri, 2002

[Irani & Basha, 2002] "*AXIS Next Generation Java SOAP*", R. Irani, S. J. Basha, Wrox Press Ltd., 2002

PuntoInformatico | Libri

WEB SERVICES IN JAVA? NON SOLO AXIS!

7.1 QUALI STRUMENTI SI HA A DISPOSIZIONE

Java nasce fin dall'inizio come linguaggio orientato al Web. È naturale che, all'affermasi delle tecnologie dei Web Sevices, la Sun, detentrice dei copyright sul linguaggio, abbia fornito una serie di strumenti per integrane l'utilizzo nelle applicazioni Java: essi sono raccolti nel Java Web Services Developer Pack (JWSDP)

7.2 JAVA WSDP

Il Java Web Services Developer Pack (JWSDP) è giunto alla versione 1.5; in tale pacchetto sono state implementate, da parte di Sun, le tecnologie relative alla gestione dei documenti XML e per i Web Services. Alla pagina *http://java.sun.com/webservices/downloads/webservicespack.html* è possibile eseguirne il download. Il download è disponibile sia per piattaforme Windows che Unix-like, e in ogni caso la dimensione è vicina ai 20 Mb.Il Developer Pack contiene classi che implementano diverse tecnologie (**Figura 7.1**), quali il parsing di documenti di tipo StAX, funzionalità per l'XML Signature, e altre tecnologie per trattare documenti XML. Vediamo i dettagli.

Figura 7.1: Contenuto di JWSDP 1.5

7.3 GESTIONE DI DOCUMENTI XML

Java possiede diverse specifiche per la gestione di documenti, tutte incluse nel JWSDP. Il primo approccio è sicuramente frastornante, visto la mole di specifiche e strumenti diversi. Inoltre alcune forniscono funzionalità non ortogonali rispetto ad altre specifiche, richiedendo una notevole conoscenza delle alternative per prendere decisioni consapevoli sugli strumenti da adottare. Per fortuna, come vedremo, non è sempre necessario conoscere tutte queste specifiche anche se, almeno a livello superficiale, è bene conoscerne l'esistenza e l'uso.

JAX-RPC: sono le Java API for XML-based Remote Procedure Call (JSR 101 e 224); la specifica definisce i mapping tra gli oggetti SOAP, WSDL, gli XSD, gli endpoint dei Web Services e gli oggetti Java corrispondenti (servizi, classi, tipi di dato del linguaggio); il supporto per mes saggi di tipo document/literal e WS-I Basic Profile è presente solo nella versione 1.1; fornisce tutti gli strumenti sia per creare client per Web Services esistenti che per creare nuovi server;
JAXB: Java Architecture for XML Binding (JSR 33, http://java.sun.com/xml/jaxb/); converte i tipi definiti in XML Schema in corrispondenti classi Java (operazione detta di marshaling); inoltre fornisce dei file XML che contengono ulteriori dichiarazioni (metadati) riguardo ai binding, per permettere una "ri-traduzione" degli oggetti in XML (operazione di un-marshaling) e più funzionalità di validazione dei documenti XML (attualmente ha un supporto non completo per RelaxNG, uno schema XML semplificato e proposto dal gruppo Oasis). È integrato con JAX-RPC per il trasporto dei messaggi. Rispetto a quest'ultimo permette, attraverso le metainformazioni aggiuntive, di avere un maggior controllo sui documenti XML generati, permettendo, per esempio, di specificare se una propriety Java diverrà un elemento o un attributo.
JAXP: Java API for XML Processing (sito di riferimento http://java.sun.com/xml/jaxp/, JSR 5 e 63); fornisce primitive per la gestione di documenti XML attraverso tool come SAX2 (si veda www.sax-

project.org/) e parser XML di tipo DOM Level 2 (www.w3.org/TR/DOM-Level-2-Core/), processori XSLT e utilità per diverse tecnologie XML (come XBase, XLink, XPath, and XPointer); supporta anche W3C XML Schema;
JAXR; Java API for XML Registries (http://java.sun.com/xml/jaxr/, JSR 93); permette di accedere a registri basati su XML (tali registri permettono di organizzare, mettere in relazione e aggiungere meta-informazioni a servizi e risorse). Fornisce API per aprire connessioni verso tali registri, interrogarli e aggiornarne i contenuti. Tra i registri supportati c'è anche la specifica UDDI 2.0 (http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm), ma anche ebXML (www.oasis-open.org/committees/tc_home.php?wg_abbrev=regrep) ed eCo Framework;
SAAJ: SOAP with Attachments API for Java; è una specifica per trattare in maniera opportuna file binary di grosse dimensioni; in pratica utilizza un formato simile agli attachment utilizzati nelle e-mail.

7.4 STAX

Tra le novità più significative c'è da segnalare la presenza di una nuova tecnologia per il parsing dei documenti XML: Sun Java Streaming XML Parser (SJSXP). Tale tecnologia fornisce un insieme di API per la gestione (intesa come operazioni di lettura e di scrittura) di documenti XML basati su stream.
Storicamente si è vista l'evoluzione delle tecnologie di parser che partivano da parser di tipo DOM (document object model) che caricavano tutto il documento in memoria e lo rendevano disponibile solamente al termine di tale operazione. Questa tecnologia mostra i suoi limiti quando i documenti sono molto grandi e il fatto di dover attendere il completamento del caricamento di tutto il documento poneva limiti di performance e di consumo eccessivo di risorse (per non parlare dei casi in cui era materialmente impossibile eseguirlo). Successivamente hanno fatto la loro comparsa i parser chiamati SAX: essi si basavano su eventi. Ogni

nuovo elemento veniva "segnalato" grazie alla generazione di un evento apposito. Veniva creata una classe per gestire un particolare evento; un insieme di classi permetteva di gestire tutti gli eventi voluti. Infine i nuovi parser, chiamati per l'appunto Streaming XML Parser e implementati con il nome di *Streaming Api for XML* (da cui la sigla StAX) prevedono la gestione a stream. La specifica delle loro funzionalità è reperibile in JSR 173, che si concretizza in un parsing di tipo "pull": infatti è il programmatore che decide quando prendere (o "tirare", per cui il verbo "pull", in inglese) il prossimo evento attraverso una qualche forma di interazione. Si passa da un ruolo passivo della tecnologia SAX (invocazione a seguito di un evento) ad un ruolo attivo (si "tira" il prossimo evento disponibile). Rispetto alla tecnologia DOM rimane meno esigente di risorse (soprattutto di memoria) e più veloce, rispetto invece alla tecnologia SAX permette la costruzione di programmi più semplici per il programmatore.
L'implementazione SJSXP di StAX non valida i documenti.

7.5 XML DIGITAL SIGNATURES

La firma digitale di documenti (nel contesto dei Web Services si parla di documenti XML) permette di verificare che il contenuto di un documento sia stato originato da una specifica fonte e non sia stato alterato durante il trasporto.
Al fine di firmare digitalmente i messaggi SOAP si utilizzano le XML Segnature. WSDP implementa la specifica JSR105, specifica che definisce come utilizzare i servizi che realizzano lo standard W3C per l'XML Segnature (www.w3.org/TR/2002/RECxmldsig-core-20020212/). Sono fornite API sia per eseguire la firma che per validare documenti firmati (maggiori informazioni sulle tecnologie legate alla sicurezza saranno fornite nel Capitolo 10).

REALIZZARE UN CLIENT NEI DIVERSI LINGUAGGI

A differenza di quanto visto nel capitolo 2, qui non verrà spiegato come si installano i singoli linguaggi utilizzati, ma si mostrerà solo il loro utilizzo. Questo perché questo capitolo, più che essere un'introduzione ai diversi linguaggi e strumenti, è un'introduzione a framework specifici utilizzati in tali linguaggi per realizzare Web Services (parte client e/o server). Pertanto solo quando è necessario eseguire un download e un'installazione specifica di framework separato dall'installazione standard saranno mostrati anche i passi per installarlo. Anche le applicazioni realizzate saranno fruibili in tutte le loro funzionalità (la maggior parte da pagine Web, altre da applicazioni stand-alone) ma si è privilegiato l'aspetto di interazione con il Web Service realizzato piuttosto che la completezza dell'interfaccia applicativa verso l'utente; per questo motivo, spesso e volentieri, si è semplificata al massimo l'interfaccia utente lasciando solo i componenti essenziali per il funzionamento dell'esempio.

8.1 VB.NET

La scelta del linguaggio, tra quelli supportati dalla piattaforma, è una semplice variante sintattica. Vista la gran quantità di sviluppatori che hanno dimestichezza con la sintassi Visual Basic, si è scelto di utilizzare il linguaggio VB.NET per l'esempio. La prima scelta è il tipo di progetto. Nell'esempio si creerà un'applicazione Windows che interagisce con il server. Pertanto si crei un progetto di Visual Basic di tipo Applicazione per Windows e lo si chiami WinAppExampleWSClient.

8.2 CREARE IL RIFERIMENTO WEB

Come fatto in precedenza, per l'esempio del Capitolo 2, è

necessario innanzi tutto creare le classi per gestire il servizio Web agendo sul menu **Progetto > Aggiungi riferimento web** e, da lì, indicare l'URL del file WSDL che descrive il servizio Web.È interessante notare la struttura creata sul file system: sotto il progetto viene creata una cartella Web References e lì vengono salvati sia il WSDL reperito che le classi VB.NET generate (**Figura 8.2**).

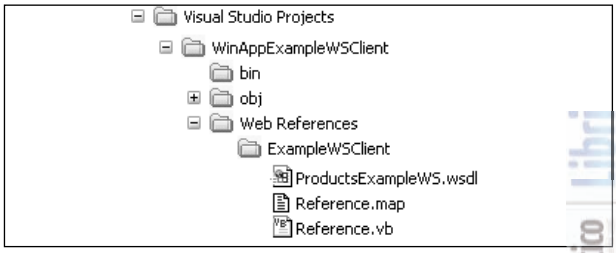

Figura 8.2: La struttura del progetto sul file system.

8.3 IL FORM

Il form del progetto permetterà all'utente di interagire con il servizio Web. Lo si imposti in maniera che sia possibile specificare una data su un casella di testo; tale data sarà il parametro dell'operazione productsOrder; un pulsante permetterà di invocare tale operazione.

Figura 8.3: Il form.

Poi è necessaria una lista di riepilogo per memorizzare i prodotti reperiti e un pulsante per effettuare l'ordine dei prodotti selezionati (si imposti la selezione multipla sulla lista, impostando SelectionMode a MultiSimple). Si può prevedere anche un pulsante Ordina e uno Annulla, inizialmente non visibili. Il form risultante è mostrato in (**Figura 8.3)**.

8.4 IL CODICE DI GESTIONE

Creato il form è necessario inserire il codice di gestione degli eventi. Per gestire l'evento di click su un pulsante è sufficiente cliccarci sopra, in modalità di progettazione del form, per vedere apparire il gestore dell'evento click Si riempia tale gestore con il seguente codice:

```
Private Sub pAggiorna_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles pAggiorna.Click
      Dim ws As New ExampleWSClient.ProductsExampleWS
      Dim ris() As ExampleWSClient.Product
      ris = ws.productsList(txtData.Text)
      portaProdotti(ris)
      pOrdina.Visible = True
End Sub
```

In pratica si crea un nuovo oggetto a partire dalla classe ExampleWSClient, creata dal WSDL: l'oggetto è di tipo ProductsExampleWS e permetterà di invocare le operazioni definite dal Web Service.
L'operazione da invocare in questo contesto è productsList, a cui si passa il valore inserito nella casella di testo, e il risultato è memorizzato in una variabile di tipo Product. Si noti che dopo l'invocazione si rende visibile il pulsante Ordina. La routine che popola la lista di riepilogo è la seguente:

```
Private Sub portaProdotti( _
  ByVal prodotti() As ExampleWSClient.Product)
    ListaProdotti.Items.Clear()
    Dim p As ExampleWSClient.Product
    For Each p In prodotti
        If p.available Then
            ListaProdotti.Items.Add( _
                myTrim(p.id, 8) + _
                myTrim(p.price, 5, False) + _
                myTrim(p.currency, 4) + _
                myTrim(p.description, 30))
        End If
    Next
End Sub
```

myTrim è una funzione che, semplicemente, imposta la dimensione della stringa passata come argomento al numero di caratteri specificato nel secondo argomento, mettendo eventuali parametri mancanti (a sinistra o a destra a seconda del valore specificato nel parametro left; se non viene specificato alcun valore left vale True). Il codice della funzione è reperibile sul CD-Rom. Si provi a testare quanto scritto finora. Il risultato dovrebbe essere come quello mostrato in (**Figura 8.4)**.

Figura 8.4: Test di invocazione dell'operazione productsList

Ora si realizzi il codice di gestione dell'evento click su Ordina. Esso eseguirà l'ordine invocando l'operazione productsOrder; dovrà visualizzare quali prodotti sono disponibili ma dovrà anche rendere visibile il pulsante Annulla e cambiare la scritta al pulsante Ordina (ora diverrà "Conferma Ordine"):

```
Private Sub pOrdina_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles pOrdina.Click
    pOrdina.Text = "Conferma ordine"
    txtData.Visible = False
    pAggiorna.Visible = False
    faiOrdine()
    pAnnulla.Visible = True
End Sub
```

FaiOrdine è la routine che legge i valori selezionati, esegue la richiesta al server e ri-popola opportunamente la lista con i prodotti effettivamente disponibili:

```
Private Sub faiOrdine()
  Dim oggetti As _
   System.Windows.Forms.ListBox.SelectedIndexCollection
  Dim ordinati() As String
  oggetti = ListaProdotti.SelectedIndices()
  Dim ordine() As ExampleWSClient.ProductInOrder
  ReDim ordine(oggetti.Count)
  ReDim ordinati(oggetti.Count)
  Dim obj As Object
  Dim i As Integer
  For i = 0 To oggetti.Count - 1
    ordine(i) = New ExampleWSClient.ProductInOrder
    ordine(i).id = prendiId(oggetti.Item(i))
    ordine(i).quantity = 1
```

```
    ordinati(i) = ListaProdotti.Items.Item( _
      oggetti.Item(i)) & " Chiesti 1 "
  Next
  Dim effettivi As ExampleWSClient.OrderWithConfirmation
  Dim ws As New ExampleWSClient.ProductsExampleWS
  effettivi = ws.productsOrder(ordine)
  ordine = effettivi.order
  txtData.Text = effettivi.magic
  ListaProdotti.Items.Clear()
  For i = 0 To ordine.Length - 1
    ordinati(i) = ordinati(i) & " ottenuti " & _
      ordine(i).quantity
    ListaProdotti.Items.Add(ordinati(i))
  Next
End Sub
```

Si noti come la quantità richiesta sia sempre 1 (per non complicare inutilmente l'interfaccia utente).
L'ID viene reperito prendendo la prima parte di stringa che compone il testo visualizzato nella lista di riepilogo:

```
Private Function prendiId(ByVal indice As Integer) As String
    Dim elem As String
    elem = ListaProdotti.Items.Item(indice)
    Dim spazio As Integer
    spazio = elem.IndexOf(" ")
    prendiId = Mid(elem, 1, spazio)
End Function
```

In (**Figura 8.5)** il risultato di una selezione e della relativa operazione di richiesta dell'ordine. Ora bisogna impostare i gestori degli eventi per i due pulsanti: il primo è sempre il pulsante Ordina. Pertanto il suo gestore deve prevedere un comportamen-

PuroInformatico Libri

to diverso in base alle due situazioni: effettua l'ordine nel caso precedente, conferma nel caso appena visto; riesce a discriminarle in base al testo visualizzato sul pulsante:

Figura 8.5: Selezione di prodotti e loro richiesta.

```
Private Sub pOrdina_Click( _
  ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles pOrdina.Click
    If pOrdina.Text = "Ordina" Then
      pOrdina.Text = "Conferma ordine"
      txtData.Visible = False
      pAggiorna.Visible = False
      faiOrdine()
      pAnnulla.Visible = True
    Else
      confermaOrdine()
      ripristinaIniziale()
    End If
  End Sub
```

Ecco le due routine utilizzate, una per confermare l'ordine:

```
Private Sub confermaOrdine()
    Dim ws As New ExampleWSClient.ProductsExampleWS
    ws.Url = MiaURL
    Dim risp As Boolean
    risp = ws.productsOrderConfirmation(txtData.Text)
    MsgBox("La conferma è stata: " & risp)
End Sub
```

L'altra per riportare la situazione allo stato iniziale. La routine ripristinaIniziale non farà altro che re-impostare i valori di default (il codice completo lo si può vedere sull'esempio allegato al CD-Rom) Infine non resta che gestire il caso in cui l'utente prema sul pulsante Annulla: non c'è altro da fare che riportarsi allo stato iniziale:

```
Private Sub pAnnulla_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles pAnnulla.Click
   ripristinaIniziale()
End Sub
```

A questo punto l'applicazione è completa e funzionante. È sempre possibile migliorarla inserendo opportune verifiche per evitare errori bloccanti (per esempio si potrebbe verificare che il valore inserito nella casella di testo sia effettivamente una data, inserire un gestore degli errori quando si tenta di accedere al servizio Web e segnalare, con messaggi non bloccanti, eventuali problemi nell'accedervi e così via).

8.5 VBA & MICROSOFT OFFICE

È possibile utilizzare una qualsiasi delle applicazioni di Office per accedere ai servizi Web e costruire un'applicazione client verso i servizi desiderati. Questo è possibile grazie ad un toolkit fornito da Microsoft (esso è gratuito e liberamente scaricabile dal-

Libri PuntoInformatico

la pagina dei download: *http://www.microsoft.com/downloads/*; basterà cercare "Microsoft Office 2003 Web Services Toolkit" per ottenere il link al prodotto). Il file da scaricare è setup.exe la cui dimensione è poco più di 800 Kb. Una volta installato il tool, basterà aprire una delle applicazioni di Office e verificare la presenza di una nuova voce di menu su *Strumenti > Web Services References* (**Figura 8.6**).

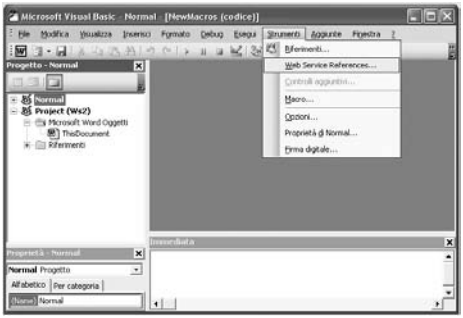

Figura 8.6: Selezione di prodotti e loro richiesta.

Selezionando tale nuova voce viene mostrato un form: esso permette sia la ricerca di nuovi Web Services utilizzando uno specifico server UDDI sia di specificare una URL da dove prelevare il WSDL che descrive il Web Service di interesse.

8.6 LE CLASSI GENERATE

In (**Figura 8.7)** si possono notare le classi create in automatico dal toolkit.La classe principale è clsws_ProductsExampleWS (infatti il tool genera sempre la classe principale con suffisso clsws_, che sta per CLaS Web Service, seguito dal nome del servizio). Il primo passo per utilizzare il servizio è dichiarare un nuovo oggetto del nuovo tipo:

```
Dim ws as New clsws_ProductsExampleWS
```

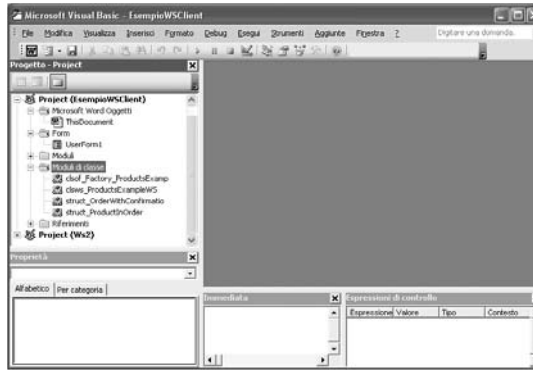

Figura 8.7: Classi generate in automatico

Su tale oggetto è possibile invocare i metodi che corrispondono alle operazioni esposte dal servizio Web di origine (per verificarlo si può ricorrere ad intelli-sense, come mostrato in **Figura 8.8**).

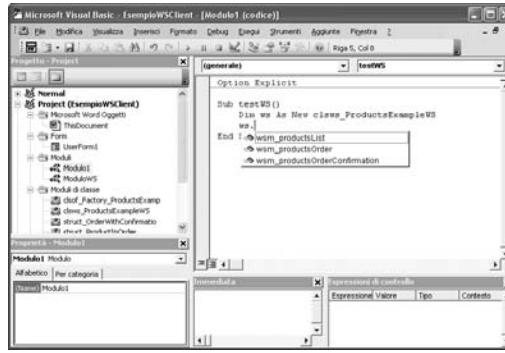

Figura 8.8: L'uso di intellisense per verificare le operazioni

Altre classi generate si riferiscono a strutture dati (iniziano con struct_, ma questo non tragga in inganno: sono a tutti gli effetti oggetti VBA) e una classe usata privatamente (Factory di oggetti di tipo struct_*).

PuntoInformatico Libri

8.7 TIPI DI DATO? NON CI SONO TUTTI!

Una cosa che si nota, verificando le classi struct_, è che manca la classe che rappresenti oggetti Product! In effetti essi vanno modellati "a mano", senza il supporto di una classe di alto livello generata automaticamente dal toolkit.In particolare è necessario utilizzare oggetti di tipo IXMLDOMSelection che modellano nodi XML. Tali nodi rappresentano i dati del documento. Ecco come stampare la lista dei prodotti restituita dal metodo productsList:

```
Sub testWS()
 Dim prodotti() As Object
 Dim prodotto As Variant
 Dim ExampleVar As New clsws_ProductsExampleWS
 prodotti = ExampleVar.wsm_productsList(Now())
 For Each prodotto In prodotti
    'Debug.Print TypeName(prodotto)
    Dim xmlProd As IXMLDOMSelection
    Set xmlProd = prodotto
    Dim i As Integer
    Debug.Print xmlProd.Context.XML
    For i = 0 To xmlProd.Length - 1
        Debug.Print "( " & xmlProd.Item(i).BaseName & ", " &
xmlProd.Item(i).Text & ")"
    Next
 Next
End Sub
```

Ora non resta che realizzare l'applicazione che fa uso delle classi generate. Si noterà che molta parte del codice è simile a quanto realizzato in VB.NET, ma si noteranno anche numerose differenze: queste sono dovute sia al diverso linguaggio sia, soprattutto, alla diversità di comportamento dei componenti visuali utilizzati

(non tanto nel modo di gestione degli eventi, ma sulle proprietà degli oggetti utilizzati).

8.8 REALIZZARE IL CLIENT

Il client che si realizza farà uso di un form. Tale form contiene sia la data da passare all'operazione per reperire i prodotti, sia una lista. Quest'ultima avrà il compito di contenere inizialmente tutti i prodotti disponibili; successivamente conterrà i prodotti ordinati (comprese le loro disponibilità). Due pulsanti permetteranno di effettuare l'ordine o di annullarlo (**Figura 8.9**).

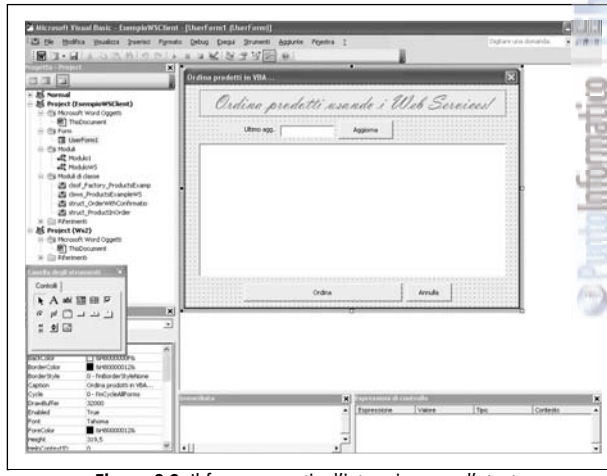

Figura 8.9: Il form per gestire l'interazione con l'utente.

Il comportamento del form è il seguente: inizialmente l'unico pulsante visualizzato è quello che permette di reperire i prodotti (pulsante Aggiorna); questo perché se non ci sono prodotti nella lista è inutile permettere all'utente di effettuare un ordine o di annullarlo! Quando l'utente preme Aggiorna, la lista viene riempita con i prodotti reperiti dal Web Service, passandogli la da-

ta specificata nella casella di testo. Contestualmente viene visualizzato il pulsante Ordina. Se l'utente lo preme, la lista viene aggiornata contenendo solo i prodotti ordinati e mostrando anche la loro disponibilità (agendo sull'operazione productsOrder del servizio Web). Inoltre Ordina cambierà etichetta diventando "Conferma ordine" e verrà visualizzato il pulsante Annulla, mentre vengono resi invisibili il pulsante Aggiorna e il campo di inserimento della data. Se l'utente conferma l'ordine verrà mostrato un messaggio che indica se la conferma è andata a buon fine (operazione productsOrderConfirmation); se invece l'utente preme Annulla l'ordine non viene confermato; in entrambi i casi ci si riporta alla situazione di partenza. Le prime routine da realizzare si riferiscono all'inizializzazione del form e all'impostazione dello stato iniziale: `UserForm_Initialize()` e `statoInizio()` (sorgenti sul CD). Successivamente si deve gestire il click sul pulsante Aggiorna:

```
Private Sub pAggiorna_Click()
   ListaProdotti.Clear
   ModuloWS.prendiProdotti CDate(txtData.Text), ListaProdotti
   statoListaProdotti
End Sub
```

Come si può notare si fa riferimento ad un modulo esterno per la gestione delle operazioni del servizio Web; si vedranno successivamente le funzionalità ivi definite. Lo stato successivo è impostato dalla seguente routine:

```
Sub statoListaProdotti()
   pOrdina.Visible = True
   pOrdina.Caption = "Ordina"
End Sub
```

A questo punto l'utente può agire sul pulsante Ordina:

```
Private Sub pOrdina_Click()
   If pOrdina.Caption = "Ordina" Then
      ' Fai ordine
      txtData.Text = ModuloWS.faiOrdine(ListaProdotti)
      statoOrdine
   End If
End Sub
```

Come si può notare viene verificata l'etichetta sul pulsante: questo perché l'azione da intraprendere varia in base allo stato attuale! Stato che ora diventa:

```
Sub statoOrdine()
   ListaProdotti.ColumnWidths = "0cm;8cm;0cm;0cm;6cm"
   pOrdina.Caption = "Conferma Ordine"
   pAnnulla.Visible = True
   txtData.Visible = False
   pAggiorna.Visible = False
   Label1.Visible = False
End Sub
```

PuntoInformatico Libri

Si noti l'uso di colonne multiple per quanto concerne ListaProdotti: solo la seconda e l'ultima colonna restano visualizzate. Ora l'utente può agire sia sul pulsante Ordina che su Annulla. Nel primo caso la procedura di gestione diventa:

```
Private Sub pOrdina_Click()
   If pOrdina.Caption = "Ordina" Then
      ' Fai ordine
      txtData.Text = ModuloWS.faiOrdine(ListaProdotti)
      statoOrdine
   Else
      ' Conferma ordine
```

```
        MsgBox "Ordine confermato? " & _
          ModuloWS.confermaOrdine(txtData.Text), , "Conferma ordine"
        statoInizio
    End If
End Sub
```

Mentre se l'utente agisce su Annulla (`routine pAnnulla_Click()`) basterà invocare la routine `statoInizio()` Ecco, nel dettaglio, il modulo con le routine che interagiscono con le classi che interrogano il Web Service (ModuloWS). prendiProdotti è la routine che si occupa di leggere la risposta di productsList (agendo su oggetti di tipo IXMLDOMSelection, come mostrato in precedenza) e di inserirla nella lista di riepilogo passata come argomento:

```
Function prendiProdotti(dt As Date, lb As ListBox) As String()
 Dim ris() As String
 Dim prodotti() As Object
 Dim prodotto As Variant
 Dim ExampleVar As New clsws_ProductsExampleWS
 prodotti = ExampleVar.wsm_productsList(dt)
 Dim numero As Integer
 numero = 0
 For Each prodotto In prodotti
    Dim xmlProd As IXMLDOMSelection
    Set xmlProd = prodotto
    If xmlProd.Item(4).Text = "true" Then
        numero = numero + 1
        ReDim Preserve ris(numero)
        Dim i As Integer
        lb.AddItem xmlProd.Item(0).Text
        For i = 1 To xmlProd.Length - 1
            lb.List(lb.ListCount - 1, i) = xmlProd.Item(i).Text
```

```
        Next
    End If
  Next
End Function
```

Ecco invece la routine che si preoccupa di eseguire l'ordine. Il parametro di ingresso indica la lista che, inizialmente, contiene la selezione dei prodotti che compongono l'ordine e, successivamente, conterrà solo i prodotti dell'ordine con le quantità disponibili:

```
Function faiOrdine(lb As ListBox) As String
    Dim ordine() As struct_ProductInOrder
    Dim numOrdinati As Integer
    Dim i As Integer
    For i = lb.ListCount - 1 To 0 Step -1
        If lb.Selected(i) Then
            'MsgBox i & " " & numOrdinati
            numOrdinati = numOrdinati + 1
            ReDim Preserve ordine(1 To numOrdinati)
            Set ordine(numOrdinati) = New struct_ProductInOrder
            ordine(numOrdinati).id = lb.List(i, 0)
            ordine(numOrdinati).quantity = 1
        Else
            lb.RemoveItem i
        End If
    Next
    Dim ExampleVar As New clsws_ProductsExampleWS
    Dim v As struct_OrderWithConfirmatio
    Set v = ExampleVar.wsm_productsOrder(ordine)
    Dim confirmed() As struct_ProductInOrder
    confirmed = v.order
    For i = lb.ListCount - 1 To 0 Step -1
```

PuntoInformatico Libri

```
        lb.List(i, 4) = "Ordinati 1, Disponibili " & _
         confirmed(i).quantity
    Next
    faiOrdine = v.magic
End Function
```

Infine ecco la routine che esegue la conferma dell'ordine:

```
Public Function confermaOrdine(magic As String) As Boolean
    Dim risp As Boolean
    Dim ExampleVar As New clsws_ProductsExampleWS
    risp = ExampleVar.wsm_productsOrderConfirmation(magic)
    confermaOrdine = risp
End Function
```

In (**Figura 8.10)** l'esecuzione dell'esempio completo.

Figura 8.10: L'esecuzione del form

8.9 PHP

PHP possiede una libreria standard per la gestione di messaggi SOAP solo a partire dalla versione 5 del linguaggio.

Per fortuna esistono numerosi pacchetti alternativi, scaricabili separatamente, disponibili anche per le versioni precedenti. In (**Tabella 8.11)** sono sintetizzate alcuni possibili package.

PACKAGE	SITO DI RIFERIMENTO
SOAP extension (PHP5)	*http://www.php.net/manual/en/ref.soap.php*
NuSOAP	*http://sourceforge.net/projects/nusoap/*
PEAR::SOAP	*http://pear.php.net/*
PHP-SOAP	*http://phpsoaptoolkit.sourceforge.net/phpsoap/*

Tabella 8.11: Toolkits per PHP

8.10 INSTALLARE PEAR:SOAP

Non sempre la procedura descritta può essere utilizzata: si pensi al caso (piuttosto comune per chi non è un programmatore PHP professionista) in cui si ha un account presso un host e non si possiede una connessione a terminale (telnet o ssh che sia), ma si ha solo la possibilità di fare upload di file via ftp. In questi casi si deve costruire "a mano" l'ambiente necessario e bisogna includere, come sottocartelle, tutte le librerie necessarie. È quanto è stato fatto per confezionare l'esempio proposto. In questo modo basterà fare l'unzip del file phpClient.zip su un sistema senza dover installare l'installer di PEAR. La struttura del client è illustrata in (**Figura 8.12)**.

Per confezionare tale esempio è stato fatto il download delle singole librerie (**Tabella 8.13**).

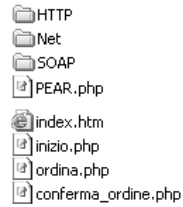

Figura 8.12: La struttura dell'esempio

LIBRERIA	URL
SOAP	*http://pear.php.net/package/SOAP*
HTTP Request	*http://pear.php.net/package/HTTP_Request*
NET Url	*http://pear.php.net/package/Net_URL*
NET Socket	*http://pear.php.net/package/Net_Socket*

Tabella 8.13: Le librerie PEAR necessarie per il client

8.11 IL CLIENT PHP PER L'ESEMPIO

Ogni pagina del client deve sia dichiarare l'inclusione del file SOAP/Client.php che prendere il WSDL del servizio per generare dinamicamente le classi; ecco come si può presentare la prima pagina di test:

```
<?php
require_once('SOAP/Client.php');
$wsdl = new SOAP_WSDL(
  'http://127.0.0.1:8080/axis/services/ProductsExampleWSPort?wsdl');
// Uno sguardo al codice generato ...
echo ( $wsdl->generateProxyCode() );
?>
```

L'invocazione di questa pagina produce il risultato mostrato in **Figura 8.14**.Una volta verificato il corretto reperimento del WSDL e la relativa generazione del proxy, si provi a invocare il metodo productsList; per farlo si aggiungano le seguenti istruzioni dopo echo:

```
$client   = $wsdl->getProxy();
$products  = $client->productsList('1980-01-01');
```

Come si può notare l'invocazione del metodo avviene impostando una data fissa, di sicuro antecedente all'inizio del servizio

Figura 8.14: il codice del proxy (generato automaticamente).

(e pertanto che permette di restituire la lista di tutti i prodotti). Per visualizzare la struttura di products basterà ricorrere all'istruzione print_r (il cui risultato è mostrato in **Figura 8.15**):

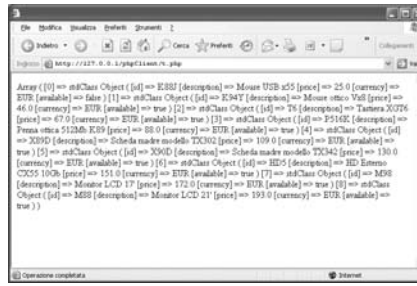

Figura 8.15: La struttura contenuta nella variabile products

```
print_r($products);
```

Purtroppo l'output è, anche in questo caso, mal formattato; ecco come risulta dopo aver aggiunto degli "a capo" opportuni (vengono mostrati solo i primi due prodotti):

```
Array (
  [0] => stdClass Object
  ( [id] => K88J
```

```
   [description] => Mouse USB x55
   [price] => 25.0
   [currency] => EUR
   [available] => false
 )
 [1] => stdClass Object
 ( [id] => K94Y
   [description] => Mouse ottico Vx8
   [price] => 46.0
   [currency] => EUR
   [available] => true
 )
 [2] => ...
)
```

Si è pronti ad utilizzare l'invocazione del metodo per costruire la prima pagina che comporrà l'applicazione client scritta in PHP! Si chiami tale pagina inizio.php e vi si inserisca il seguente codice:

```
<?php
require_once('SOAP/Client.php');
$wsdl=newOAP_WSDL('
http://127.0.0.1:8080/axis/services/ProductsExampleWSPort?wsdl');
$client   = $wsdl->getProxy();
$products  = $client->productsList('1980-01-01');
?>
<HTML>
 <HEAD>  <meta author="Filippo Costalli" /> </HEAD>
<BODY>
 <FORM NAME ="orderForm" ACTION="ordina.php" METHOD="post">
  <H3> Lista prodotti disponibili </H3>
    <TABLE border = "1">
```

```
<?
    foreach ($products as $hit)
    {
        $productID = $hit->id;
        $desc      = $hit->description;
        $price     = $hit->price;
        $currency  = $hit->currency;
        $available = $hit->available;
        echo("<TR>");
        echo("<TD>".$productID."</TD>");
        echo("<TD>".$desc."</TD>");
        echo("<TD>".$price."</TD>");
        echo("<TD>".$currency."</TD>");
        echo("<TD>".$available."</TD>");
        $order = "<TD> </TD>";
        if ($available == "true")
        {
         $order = "<TD>"
        ."<INPUT TYPE ='hidden' NAME = '".$productID."' VALUE='no'/>"
          ."<INPUT TYPE = 'CHECKBOX'"
          ."onClick =\"this.form.".$productID.".value='ok';\"</TD>";
        }
        echo($order);
        echo("</TR>");
    }
?>
 </TABLE>
 <BR/><BR/>
 <INPUT TYPE ="submit" VALUE = "Invia ordine"/>
 </FORM> </BODY> </HTML>
```

Come si può notare dal codice, ogni prodotto viene stampato su una tabella, incluso un campo di selezione. Un possibile esempio di visualizza-

PuntoInformatico Libri

zione della pagina è mostrato in (**Figura 8.16)**. Quando l'utente preme su "Invia ordine", viene fatto il submit sulla pagina ordina.php che contiene una parte che reperisce i prodotti selezionati ed esegue la chiamata al servizio Web con l'operazione productsOrder:

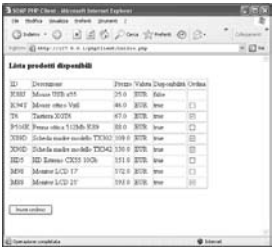

Figura 8.16: La prima pagina del client

```
<?php
require_once('SOAP/Client.php');
// Mette nell'array gli id dei prodotti ordinati
$ordini = array();
$i = 0;
foreach($HTTP_POST_VARS as $key => $value)
{
    if ($value=='ok')
    {
    $ordine = array(
        'id' => $key,
        'quantity' => 1
        );
        $ordini[$i] = $ordine;
        $i++;
    }
}
$wsdl=new SOAP_WSDL(
 'http://127.0.0.1:8080/axis/services/ProductsExampleWSPort?wsdl');
```

```
$client   = $wsdl->getProxy();

$ordineFatto  = $client->productsOrder($ordini);
$order = $ordineFatto["order"];
$magik = $ordineFatto["magic"];
?>
```

Una seconda parte, subito dopo la prima, mostra i prodotti effettivamente disponibili e chiede all'utente di confermare l'ordine (**Figura 8.17**):

Figura 8.17: La pagina che mostra i prodotti disponibili.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<BODY>
 <FORM NAME ="orderForm" ACTION="conferma_ordine.php"
METHOD="post">
 <H3> Lista prodotti ordinati </H3>
  <TABLE border = "1">
  <?
     foreach ($order as $hit)
     {
```

```
        $id       = $hit->id;
        $quantity = $hit->quantity;
        echo("<TR>");
        echo("<TD>".$id."</TD>");
        echo("<TD>".$quantity."</TD>");
        echo("</TR>");
      }
?>
    </TABLE>
    <BR/><BR/>
     <INPUT TYPE ="hidden" NAME="magic" VALUE =
"<?=$magik?>"/>
     <INPUT TYPE ="submit" VALUE = "Invia Conferma ordine"/>
   </FORM>
</BODY> </HTML>
```

Ecco l'ultima pagina, quella che conferma l'ordine e mostra il risultato di tale conferma (conferma_ordine.php), come illustrato dalla (**Figura 8.18)**:

Figura 8.18: L'ordine è stato confermato!

```
<?phprequire_once('SOAP/Client.php');
// Mette nell'array gli id dei prodotti ordinati
$magic = $HTTP_POST_VARS["magic"];
```

```
$wsdl=new SOAP_WSDL(
  'http://127.0.0.1:8080/axis/services/ProductsExampleWSPort?wsdl');
$client   = $wsdl->getProxy();
$ordineConfermato  = $client->productsOrderConfirmation($magic);
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>  <meta author="Filippo Costalli" /> </HEAD>
<BODY>
<H3> Risultato ordine: </H3>
<?php
 if ($ordineConfermato=='false')
     echo("Spiacenti, ordine inevaso");
 else
     echo("Ordine effettuato");
?>
</BODY> </HTML>
```

PuntoInformatico Libri

8.12 CONSIDERAZIONI FINALI SUL PHP

Quanto è stato illustrato coincide con la modalità di scrittura del client: dapprima sono stati verificati il corretto reperimento del file WSDL e la generazione del proxy; poi è stato analizzato il codice del proxy (attraverso il suo dump a video con echo) ed infine l'invocazione dell'operazione e la stampa del tipo di dati restituita grazie alla funzione print_r. Tutto questo è dovuto al fatto che, a differenza dei linguaggi incontrati in precedenza, il PHP genera tutte le classi del proxy dinamicamente (spesso si dice "on the fly"): questo comporta anche un processo interattivo esso stesso dinamico e quasi "try & error". Ciò non toglie l'estrema flessibilità del linguaggio (e della libreria) nonché la semplicità del suo utilizzo.

Il PHP non è l'unico linguaggio a possedere queste caratteristiche. Anche il "fratello maggiore" Perl gli è, per molti versi, simile (ma ancor più duttile per un'infinità di motivi che, in questa sede, non è il caso di approfondire) ma pure il Python (non per nulla i tre vegono indicati anche come "P-Languages" per indicarne le forti somiglianze!).
Vediamo come costruire un client usando il Perl…

8.13 PERL & SOAP:LITE

SOAP::Lite è considerato il tool più stabile disponibile per il Perl. Il sito di riferimento è *http://soaplite.com/*, mentre la mailing list è raggiungibile all'indirizzo *http://groups.yahoo.com/group/soaplite*. Anche questa libreria, com'è accaduto per la libreria SOAP del PHP, permette di costruire dinamicamente un proxy client verso un Server SOAP descritto da un opportuno WSDL. Procediamo con ordine: per prima cosa è necessario avere la libreria installata. Gli utenti windows che utilizzano la distribuzione ActivePerl (sito *http://www.activestate.com/Products/ActivePerl*) possono verificare che essa è già presente nei pacchetti standard.). Una cosa estremamente interessante (e utile!) è che SOAP:Lite è accessibile anche attraverso COM: in pratica è possibile utilizzarne le funzionalità anche all'interno dei linguaggi che prevedono l'uso di tale modello (compresi Visual Basic e pagine ASP).

8.14 ACCEDERE AD UN SERVIZIO DI ESEMPIO

Si supponga di voler accedere al Web Service di esempio primoWSEsempio, definito nel Capitolo 6; la prima cosa è dichiarare l'uso della libreria SOAP:Lite, poi specificare l'indirizzo dell'endpoint usando proxy, invocare il metodo (passandogli eventuali parametri) e prendere il risultato con result; ecco un semplice client che invoca l'operazione Metodo2 sul WS:

```
use SOAP::Lite;
print SOAP::Lite
 service('http://127.0.0.1:8080/axis/services/primoWSEsempio?wsdl')
  -> Metodo2('Un esempio in Perl');
```

Il client è bell'e pronto! Il risultato della sua esecuzione è:

```
Metodo2 risponde a Un esempio in Perl
```

8.15 ACCESSO AL SERVIZIO "REALE"

Non resta che procedere in maniera analoga per accedere al servizio di gestione degli ordini di prodotti:

```
use SOAP::Lite;
print SOAP::Lite
  ->service('http://127.0.0.1:8080/axis/services/
      ProductsExampleWSPort?wsdl')
  -> productsList('2000-01-01');
```

Purtroppo questa chiamata non restituisce alcun risultato. Per comprendere cosa accade "dietro le quinte" si può far modificare la prima riga in:

```
use SOAP::Lite +trace => qw (debug);
```

L'esecuzione ora mostra sia il messaggio SOAP generato che la risposta; ecco come appare (dopo aver modificato l'indentazione dell'output per renderlo più comprensibile):

```
SOAP::Transport::HTTP::Client::send_receive:
  POST http://127.0.0.1:8080/axis/services/ProductsExampleWSPort
Accept: text/xml
Accept: multipart/*
```

```
Content-Length: 476
Content-Type: text/xml; charset=utf-8
SOAPAction: "#productsList"
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/1999/
          XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/">
<SOAP-ENV:Body>
  <productsList>
    <c-gensym3 xsi:type="xsd:string">2000-01-01</c-gensym3>
  </productsList>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fin qui i messaggi si riferiscono alla chiamata del client. Si noti come, evidenziato in grassetto, viene generato un messaggio SOAP con stile RPC. Segue la risposta del server:

```
SOAP::Transport::HTTP::Client::send_receive: HTTP/1.1 500 Internal Server Error
Connection: close
Date: Sat, 16 Apr 2005 08:26:21 GMT
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Client-Date: Sat, 16 Apr 2005 08:26:21 GMT
Client-Peer: 127.0.0.1:8080
Client-Response-Num: 1
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="
      http://schemas.xmlsoap.org/soap/envelope/"
```

```
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
 <soapenv:Fault>
  <faultcode>soapenv:Server.userException</faultcode>
  <faultstring>
   org.xml.sax.SAXException: SimpleDeserializer encountered
          a child element,
   which is NOT expected, in something it was trying to deserialize.
  </faultstring>
  <detail>
   <ns1:hostname xmlns:ns1="http://xml.apache.org/axis/">
    9agosto2003
   </ns1:hostname>
  </detail>
</soapenv:Fault>
</soapenv:Body>
</soapenv:Envelope>
```

Una soluzione a questo problema è scrivere il messaggio SOAP in questo modo:

```
#!/usr/bin/perl -w
use strict;
use LWP::UserAgent;
use HTTP::Request::Common;
my $proxy = "http://127.0.0.1:8080/
          axis/services/ProductsExampleWSPort";
my $uri='http://ivenuti.altervista.org/ProductsExampleWS.xsd1';
my $action = "$uri/productsList";
my $date = "2005-04-11";
my $userAgent = LWP::UserAgent->new(agent => 'Perl SOAP');
my $message = "<?xml version=\"1.0\" encoding=\"utf-8\"?>
```

unitoInformatico Libri

```
<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">
<soapenv:Body>
<ElementDatexmlns=\"http://ivenuti.altervista.org/
            ProductsExampleWS.xsd1\">$date</ElementDate>
</soapenv:Body>
</soapenv:Envelope>";
my $response = $userAgent->request(POST $proxy,
Content_Type => 'text/xml',
SOAPAction => $action,
Content => $message);
print $response->error_as_HTML unless $response->is_success;
print $response->as_string;
```

Si è dovuto far ricorso ad una programmazione a basso livello, componendo i messaggi pezzo a pezzo! Per fortuna è possibile internevire sul file WSDL al fine di renderlo maggiormente adatto anche a questo tool. I dettagli verranno illustrati nel Capitolo 9.

8.16 PYTHON

Python possiede due librerie "stabili" per la gestione di messaggi SOAP: SOAPpy e ZSI. Si illustrerà l'utilizzo si SOAPpy.

8.17 DOWNLOAD E INSTALLAZIONE DI SOAPPY

La versione disponibile della libreria, al momento di scrivere il libro, è la 0.11.6. Essa è scaricabile dal sito *http://pywebsvcs.sf.net/*. Essa è una libreria per scrivere client e server per i Web Services.

Anche in questo caso (come accaduto per il Perl e la libreria SOAP::Lite) ci sono problemi a realizzare in automatico client verso servizi document/literal.
Come per il Perl, e come suggerito anche nella pagina *http://users.skynet.be/pascalbotte/rcx-ws-doc/postxmlpython.htm)* non resta che realizzare "a mano" l'invocazione del metodo:

```
import sys, httplib
plainMsg = """<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<ElementDatexmlns="http://ivenuti.altervista.org/
        ProductsExampleWS.xsd1">%s</ElementDate>
</soapenv:Body>
</soapenv:Envelope>
"""
SOAPMsg = plainMsg%("2000-01-01")
service = httplib.HTTP("127.0.0.1:8080")
service.putrequest("POST", "/axis/services/ProductsExampleWSPort")
service.putheader("Host", "127.0.0.1:8080")
service.putheader("User-Agent", "Python example")
service.putheader("Content-type", "text/xml; charset=\"UTF-8\"")
service.putheader("Content-length", "%d" % len(SOAPMsg))
service.putheader("SOAPAction",
 "http://ivenuti.altervista.org/ProductsExampleWS.xsd1/productsList")
service.endheaders()
service.send(SOAPMsg)
statuscode, statusmessage, header = service.getreply()
print "Response: ", statuscode, statusmessage
```

PuntoInformatico Libri

```
print "headers: ", header
res = service.getfile().read()
print res
```

Si rimanda al Capitolo 9 per la ricerca di una soluzione alternativa, più semplice di questa: per farlo sarà necessario modificare il WSDL.

8.18 RISULTATI OTTENUTI? NON QUELLI SPERATI!

In conclusione a questo Capitolo non si può non sottolineare che i risultati ottenuti siano modesti. È vero che in quasi tutti i linguaggi si è potuto realizzare un client, ma in molti casi quanto è stato scritto è di livello troppo basso per essere considerato utile per un programmatore medio, che ha poca conoscenza dei dettagli specifici dei Web Services. Per fortuna è possibile intervenire sul WSDL per migliorare la situazione. Si tenga conto che non sempre è possibile intervenire sul WSDL: spesso questo è pubblicato da terze parti che non hanno interesse a modificarlo. Tale modifiche sarebbero critiche per tutti i client precedentemente realizzati. Ecco perché è importante che un WSDL sia testato e verificato su diversi client prima di procedere alla sua pubblicazione definitiva!

BIBLIOGRAFIA

[MSDNInterop] "*Web Services Interoperability*", http://msdn.microsoft.com/webservices/building/interop/
[PHPClient] "*A PHP Web Services Client*", A. Trachtenberg, http://www.onlamp.com/lpt/a/3968
[SOAPLite] http://cookbook.soaplite.com/

PuntoInformatico | Libri

CLIENT E SERVIZI CON MESSAGGI SOAP WRAPPED/LITERAL

Si è potuto osservare, nel capitolo precedente, come alcune librerie fallissero nel creare dei client per accedere ai servizi Web come sono stati esposti da Axis. L'origine è dovuta ad una gestione non ottimale del formato document/literal da parte dei tool. Esiste un formato di messaggi SOAP più vicino allo stile RPC, ma ancora di tipo Document: è lo stile wrapped.

9.1 RIVISITARE IL WSDL ORIGINARIO

Se si vuole mantenere una certa "similitudine" con lo stile rpc, è bene usare la forma wrapped, il che significa che il documento WSDL deve soddisfare queste ulteriori regole:

1. Nella sezione message, tutti i messaggi di input devono avere una parte singola;
2. Tale parte è un element;
3. Il nome dato ad element è lo stesso dell'operazione;
4. Ogni element deve contenere un complex type senza attributi

In pratica bisogna eseguire le seguenti sostituzioni (per parole intere!) sul documento originario generato secondo le indicazioni del Capitolo 3: Inoltre è opportuno modificare il tipo nonNegativeInteger al fine di rendere il WSDL utilizzabile anche da Axis C++; tra i tipi di dato supportati, e adatti al nostro scopo, c'è unsignedInt. Se si eseguono delle prove si vedrà che tale tipo non è supportato da un client scritto in Python con SOAPpy; non resta che utilizzare un semplice int e verificare, sul server, che la quantità ordinata sia maggiore di zero (se non lo è basta rispondere con un numero di prodotti prenotati pari a 0).

9.2 UN CLIENT PERL

Come descritto nel Capitolo precedente si farà uso della libreria SOAP::Lite. Per rendere il client accessibile via Web, si procederà alla creazione di uno script CGI-BIN; ecco la procedura principale:

```
#!c:\perl\bin\perl.exe
# autori: gli amici di perl.it
use strict;
use warnings;
use CGI;
use CGI::Carp qw(fatalsToBrowser);
use SOAP::Lite;
our ($service,$query);
MAIN: {
  my $soap = new SOAP::Lite;
  $service=$soap->
     uri('http://127.0.0.1:8080/axis/services/
            ProductsExampleWSWrappedPort')
     ->proxy('http://127.0.0.1:8080/axis/services/
            ProductsExampleWSWrappedPort')
                  or die 'Problemi al web service';
  $query = new CGI;
  print $query->header,
    $query->start_html("SOAP Perl client");
  if ($query->param('conferma')) {
    conferma_ordine();
  } elsif ($query->param('ordina')) {
    ordina();
  } else {
    mostra_prodotti();
  }
  print $query->end_html;
}
```

PuntoInformatico Libri

Ora è necessario implementare le diverse routine; la prima è quella che mostra i prodotti reperiti dal Web Service (la data è resa fissa):

```
sub mostra_prodotti {
 my $ris=$service->productsList(
        SOAP::Data->name(
          'updateDate' =>SOAP::Data->type('date'=>'1970-01-01')
          )
    );
 my @rows=$query->td(['','Prodotto','Prezzo','Valuta']);
 print $query->start_form;
 foreach my $elem ($ris->valueof( '//item' )) {
   my $value=$elem->{'id'};
   if ($elem->{'available'} ne 'false') {
     push @rows, $query->td([
       $query->checkbox(-name=>'items',
         -value=>$elem->{'id'},
         -label=>''),
       $elem->{'description'},
       $elem->{'price'},
       $elem->{'currency'}
     ]);
   } else {
     push @rows, $query->td([
       '',
       $elem->{'description'},
       $elem->{'price'},
       $elem->{'currency'}
     ]);
   }
 }
 print $query->table({border=>1},$query->Tr(\@rows)),
```

```
  $query->br,
  $query->submit('ordina','Invia ordine'),
  $query->end_form;
}
```

Ecco la procedura che effettua l'ordine (per i soli prodotti selezionati dall'utente):

```
sub ordina {
 my @orders;
 my @items=$query->param('items');
 my @rows=$query->td(['Codice prodotto','Quantità']);
 foreach my $id (@items) {
   push @orders, SOAP::Data->name(
       'item' => SOAP::Data->type(
           'ordered_hash' =>[ id => $id, quantity => 1]
         )
       );
 }
 my $reply = $service->productsOrder(@orders);
 foreach my $item ($reply->valueof( '//order/item' )) {
   push @rows, $query->td([
         $item->{ 'id' } , $item->{ 'quantity' }
       ]);
 }
 print $query->table({border=>1},$query->Tr(\@rows)),
       $query->br,
       $query->start_form,
       $query->hidden('magic',$reply->valueof( '//magic' )),
       $query->submit('conferma','Conferma ordine'),
       $query->end_form;
}
}
```

PuntoInformatico Libri

Infine ecco come potrebbe essere scritta la procedura che conferma l'ordine con le quantità effettivamente rese disponibili:

```
sub conferma_ordine {
 my $magic=$query->param('magic');
 my $order=$service->productsOrderConfirmation(
      SOAP::Data->name(
        'productsOrderConfirmation' =>SOAP::Data->type('
            string'=>$magic)
      )
    );

 if ($order->valueof( '//fl' ) eq 'false') {
   print 'Ordine NON confermato';
 } elsif ($order->valueof( '//fl' ) eq 'true') {
   print 'Ordine confermato';
 } else {
   print 'Errore nella transazione';
 }
```

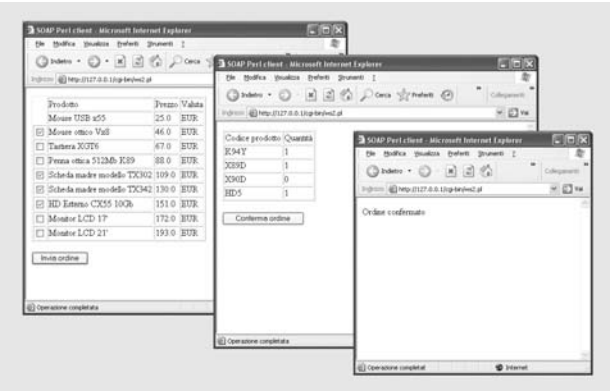

Figura 9.1: Esecuzione del client scritto in Perl

Per la sua esecuzione, se si dispone del Web Server Apache, basterà salvare il file nella cartella cgi-bin, far partire Apache e invocare l'url *http://127.0.0.1/cgi-bin/nomeScript.pl*. In (**Figura 9.1)** un esempio di esecuzione dello script proposto.

9.3 UN CLIENT GRAFICO? SÌ E ANCHE MULTIPIATTAFORMA CON WX!

Benché il Perl trovi il suo "naturale" utilizzo come linguaggio per la scrittura di cgi-bin, è possibile utilizzarlo anche per scrivere applicazioni. In questo caso è possibile realizzare l'interfaccia non più con codice Html, ma attraverso una GUI. Esistono numerose librerie grafiche: per esempio si può utilizzare WX, il cui sito di riferimento è http://wxperl.sourceforge.net. WX permette tra le altre cose, di realizzare interfacce multipiattaforma. In (**Figura 9.2)** un esempio di esecuzione di un'applicazione realizzata da Stefano Rodighiero (larsen@perl.it) e resa disponibile per il download all'indirizzo http://larsen.perlmonk.org/src/wspe.tar.gz..
La figura mostra l'esecuzione di una stessa operazione ma eseguita su sistemi diversi (Windows, Linux e Macintosh).

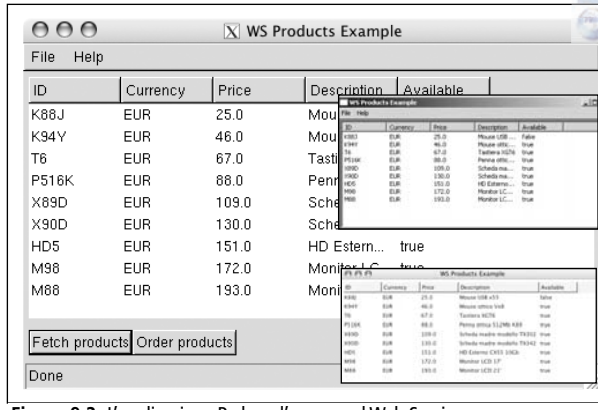

Figura 9.2: L'applicazione Perl per l'accesso al Web Service (in Windows, Linux e Macintosh!)

9.4 PYTHON

Il client scritto in Python non dà problemi né per quanto concerne la lista dei prodotti (operazione productsList) né per la conferma di un ordine (productsOrderConfirmation); invece è problematica la gestione di un array di prodotti per eseguire l'ordine (productsOrder). Però si può sempre utilizzare un semplice accorgimento: anziché effettuare l'ordine di un array di prodotti, si eseguono tanti ordini quanti sono i prodotti, ordinandone uno per volta! Ovviamente l'eventuale conferma (dopo aver verificato che la disponibilità è maggiore di 0) deve essere fatta subito. Ecco un esempio di client che, reperita la lista dei prodotti ordinabili (passando come argomento la data del sistema) ordina ogni prodotto (quantità 1) ed esegue la conferma se tale prodotto è disponibile (un possibile risultato è mostrato in **Figura 9.3**):

```
C:\WINDOWS\system32\cmd.exe

D:\IVAN\scritti\libri\ws\esempi\pythonClients>ExampleWStestWrapped2.p

-----<< Client SOAPpy >>----------------------------------------

Disponibile il prodotto 'Scheda madre modello TX302', id=X89D
num prodotti disponibili=1
prodotto ordinato con magic=TK0.936015635925657, risposta=true

Disponibile il prodotto 'Penna ottica 512Mb K89', id=P516K
num prodotti disponibili=1
prodotto ordinato con magic=TK0.3743256316583188, risposta=true

Disponibile il prodotto 'Tastiera XGT6', id=T6
num prodotti disponibili=0
Prodotto non ordinabile...

Disponibile il prodotto 'Mouse ottico Vx8', id=K94Y
num prodotti disponibili=1
prodotto ordinato con magic=TK0.3675311786713894, risposta=true

Disponibile il prodotto 'Mouse USB x55', id=K88J
num prodotti disponibili=0
Prodotto non ordinabile...


-----<< Fine >>-------------------------------------------------


D:\IVAN\scritti\libri\ws\esempi\pythonClients>_
```

Figura 9.3: Un esempio di esecuzione del client SOAPpy

```
#!/usr/bin/env python
import SOAPpy
import datetime
import calendar
```

```
from SOAPpy import *
from datetime import date
# Prende la lista dei prodotti
print
print "———<< Client SOAPpy >>————————————————————"
print
server = SOAPProxy(
  'http://127.0.0.1:8080/axis/services/
            ProductsExampleWSWrappedPort?wsdl',
  namespace = 'urn:http://ivenuti.altervista.org/
            ProductsExampleWSWrapped.xsdl')
data = date.today()
prodotti = server._ns(
  "http://ivenuti.altervista.org/
ProductsExampleWSWrapped.xsdl").productsList(data)
for x in range(len(prodotti)):
  #Prende il prossimo prodotto
  prodotto = prodotti.pop()
  print "Disponibile il prodotto '"+prodotto['description']+
            "', id="+prodotto['id']
  ordine = {}
  ordine['id'] = prodotto['id']
  ordine['quantity'] = 1
  #Lo prenota
  prenotati = server._ns(
   "http://ivenuti.altervista.org/
      ProductsExampleWSWrapped.xsdl").productsOrder(
     item = ordine)
  token = prenotati['magic']
  print "num prodotti disponibili="+prenotati['order']['item']['quantity']
  #Se quantity>0 (prodotto disponibile) esegue la conferma
  if (prenotati['order']['item']['quantity'])>'0':
      risposta = server._ns(
```

PuntoInformatico Libri

```
        "http://ivenuti.altervista.org/ProductsExampleWSWrapped.xsdl"
        ).productsOrderConfirmation(magic=token)
      print "prodotto ordinato con magic="+token+", risposta="+risposta
  else:
      print "Prodotto non ordinabile..."
  print
print
print "———<< Fine >>————————————————————————————————"
print
```

9.5 ALCUNI PROBLEMI "APERTI"

Ci sono alcuni casi in cui è obiettivamente difficile garantire soluzioni portabili. La rappresentazione di date può essere problematica: Java utilizza informazioni relative ai fusi orari delle diverse località, la piattaforma .NET no. Che dire della rappresentazione di date "nulle"? Alcuni linguaggi considerano tali date come date minime rispetto ad un range prefissato, altri usano stringhe vuote, altri ancora costrutti particolari (null, nil, e così via).
Chi sviluppa in Java deve anche porsi il problema di come eventuali strutture dati complesse (hashtable, hashmap e altre) vengano gestite dai Web Services: il rischio è che si debba ricorrere a soluzioni ad hoc per la loro gestione, con ripercussioni negative sull'interoperabilità.
Alcuni sono critici anche rispetto all'interoperabilità tra diversi toolkit riguardo al formato document/literal; a tal proposito si veda il parere di Nelson Minar di Google basato sulla propria esperienza (dalla pagina *http://www.windley.com/archives/2005/03/nelson_minar_at.shtml*): secondo lui solo Java (con Axis) e .Net hanno un buon livello di interoperabilità.

9.6 CONSIDERAZIONI FINALI

Si è potuto verificare, con un esempio concreto, come l'interoperabilità sia tutt'ora un problema. Non basta far sì che il proprio WSDL soddisfi lo

standard WS-I Basic Profile, ma è necessario realizzare i client nei linguaggi in cui si vuole garantire la compatibilità e verificare in concreto l'interoperabilità. Questo si traduce senz'altro in uno sforzo notevole nella predisposizione di un ambiente iniziale e, probabilmente, nella necessità di rivisitare il WSDL prima di pubblicarlo "ufficialmente". Però questo sforzo viene senz'altro ripagato in termini di possibili "clienti": maggiori saranno i linguaggi (con i relativi tool) utilizzabili, minori i vincoli da imporre a chi vuole utilizzare il proprio servizio. Questo è sicuramente un ottimo "biglietto da visita" e un ottimo indice di qualità della soluzione proposta.Come accennato nei primi capitoli è importante garantire che il WSDL, una volta pubblicato, non venga modificato, essendo un vero e proprio contratto con il mondo esterno.

PuntoInformatico | Libri

INDICE

Introduzione 3

Architetture Distribuite
1.1 Client-Server 6
1.2 Tre Livelli (o più) 7
1.3 Service Oriented Architecture (SOA) 8
1.4 Cosa sono i Web Services? 11
1.5Perchè usare i Web Services? 12
1.6 Semplicità della Specifica 13
1.7 Utilizzo di Tecnologie Standard 14
1.8 Ampio Consenso da parte di diverse Aziende 14
1.9 Presenza di Tool 15
1.10 Non è tutto oro... 15

Il primo Web Services in...un Battibaleno
2.1 Un Web Service Semplice Semplice 17
2.2 Installare Axis, Tomcat e il Java Developer Kit 17
2.3 Il Primo Web Service? Una Classe Java 21
2.4 Ottenere il..."Contratto" 22
2.5 Realizzare il Client in VB.NET 23
2.6 Tutto Qui? 27

Tecnologie Sottostanti i Web Services
3.1 Internet e lo Stack OSI 29
3.2 Xml e Tecnologie Connesse 30
3.3 Attributi Xml 32
3.4 Attributi o Elementi? 32
3.5 Xml Namespace 33
3.6 Xml Schema e Dtd 34
3.7 Xml Schema per la Definizione 34
3.8 Costruire un Xml Schema 37

3.9 Soap, Wsdl e Uddi 40
3.10 SOAP 40
3.11 I Messaggi 40
3.12 Il problema del Trasporto (HTTP, SMTP, ...) 41
3.13 Il problema della Sicurezza 42
3.14 Scelte Architetturali 42
3.15 WSDL 44
3.16 TYPES 46
3.17 MESSAGE 46
3.18 PORTTYPE 47
3.19 BINDING 47
3.20 SERVICE 47
3.21 I Formati del Messaggio 48
3.22 L'uso: Encoded o Literal 48
3.23 UDDI 49
3.24 Conclusioni 49

Collaborare? WS-I Basic Profile

4.1 Problemi di Compatibilità 54
4.2 WS-I BASIC PROFILE in dettaglio 54
4.3 Requisiti di Conformità 55
4.4 Punti di Estensione 55
4.5 Conformance Target 55
4.6 Alcune Caratteristiche 56
4.7 WS-I Testing Tools 57
4.8 Download e Installazione 57
4.9 File di Configurazione di Analyzer 58
4.10 Esecuzione di Analyzer 58
4.11 Configurazione e Analisi su primoWS 59

Realizzare un WSDL Partendo da Zero

5.1 L'applicazione di Esempio 63
5.2 Prima l'uovo o la Gallina? 64

PuntoInformatico | Libri

5.3 Le Regole da Seguire 65
5.4 WSDL e XML Schema con SOA Editor 65
5.5 Definire le Operazioni 66
5.6 Definire i Tipi 69
5.7 Definire i Messaggi 77
5.8 Bindings 79
5.9 Services 79
5.10 Test di Conformità 83

Axis in Dettaglio

6.1 Apache Web Services Project 85
6.2 Un Esempio un pò più Complesso 85
6.3 La Configurazione via WSDD 86
6.4 Scrivere il client 88
6.5 Ulteriori Opzioni: lo Scope 89
6.6 Usare AdminClient 90
6.7 L'Architettura 93
6.8 Un Esempio di Handler 95
6.9 Come Inserirlo in una Catena 97
6.10 Usare Handlers o Filtri J2EE 96
6.11 Inserire un Proprio WSDL 96
6.12 Gli "Stili" Disponibili in Axis 99
6.13 Come Utilizzare Axis 99
6.14 Tool "Esterni" 98
6.15 Realizzare un Client! 98
6.16 I Messaggi tra Client e Server 101

Web Services in Java? Non solo Axis

7.1 Quali Strumenti si ha a Disposizione? 107
7.2 Java WSDP 107
7.3 Gestione di Documenti XMl 108
7.4 STAX 109
7.5 XML Digital Signatures 110

Realizzare un Client nei Diversi Linguaggi

8.1 VB.NET 111
8.2 Creare il Riferimento Web 111
8.3 Il Form 112
8.4 Il Codice di Gestione 113
8.5 VBA & Microsoft Office 118
8.6 le Classi Generate 119
8.7 Tutti i tipi di Dato? Non ci sono Tutti! 121
8.8 Realizzare il Client 122
8.9 PHP 127
8.10 Installare Pear:Soap 128
8.11 Il Client PHP per l'Esempio 129
8.12 Considerazioni Finali sul PHP 136
8.13 Perl & Soap:Lite 137
8.14 Accedere ad un Servizio di Esempio 137
8.15 Accesso al Servizio "Reale" 138
8.16 Python 141
8.17 Download e Installazione di Soappy 141
8.18 Risultati Ottenuti? Non Quelli Sperati! 143

Client e Servizi con Messaggi Soap Wrapped/Lieral

9.1 Rivisitare il WSDL Originario 145
9.2 Un Client Perl 146
9.3 Un Client Multipiattaforma 150
9.4 Python 151
9.5 Alcuni Problemi "Aperti" 153
9.6 Considerazioni Finali 153

PuntoInformatico | Libri

i libri di
ioPROGRAMMO

WEB SERVICES IN PRATICA

Autore: Ivan Venuti
Responsabile Editoriale: Gianmarco Bruni

EDITORE
Edizioni Master S.p.A.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona ind. - 87036 Rende (CS)

Realizzazione grafica:
Cromatika Srl
C.da Lecco, zona ind. - 87036 Rende (CS)

Resp. grafico: Paolo Cristiano
Coordinatore tecnico: Giancarlo Sicilia
Illustrazioni: Mario Veltri
Impaginazione elettronica: Francesco Maddalone

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Servizio Clienti 02/831212

Stampa: Grafica Editoriale Printing - Bologna

Finito di stampare nel mese di Ottobre 2005

Il contenuto di quest'opera, anche se curato con scrupolosa attenzione, non può comportare specifiche responsabilità per involontari errori, inesattezze o uso scorretto. L'editore non si assume alcuna responsabilità per danni diretti o indiretti causati dall'utilizzo delle informazioni contenute nella presente opera. Nomi e marchi protetti sono citati senza indicare i relativi brevetti. Nessuna parte del testo può essere in alcun modo riprodotta senza autorizzazione scritta della Edizioni Master.

Copyright © 2005 Edizioni Master S.p.A.
Tutti i diritti sono riservati.